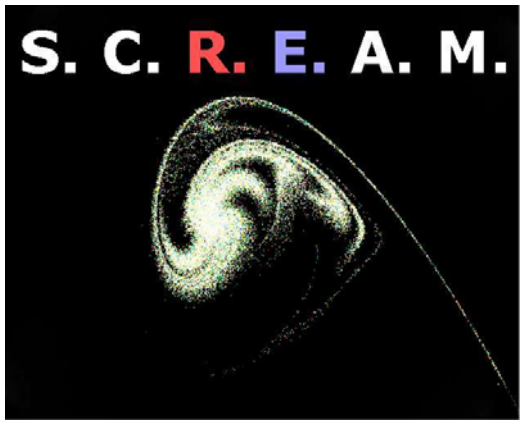
Fermi National Accelerator Laboratory
PO Box 500
Batavia, IL 60510
TD-04-031
May 2005



Superconducting Relativistic Particle Accelerator Simulation

A program description

P. Bauer[1], G.W. Foster, M. Huening

Abstract --- The following note describes a computer code used for the simulation of the acceleration of a pulsed beam through a linac composed of superconducting cavity resonators. This program was developed by M. Huening for the simulation of the longitudinal beam dynamics in the Tesla Test Facility (TTF) at DESY. The inner workings of the different modules composing the S.C.R.E.A.M program are described, including a detailed discussion of the program input, commented flowcharts and a detailed documentation of the algorithm. The program is now being used for the simulation of the proposed Fermilab Proton-Driver.

[1] pbauer@fnal.gov

TABLE OF CONTENT

1	SUMMARY	1- 3
2	INTRODUCTION	2- 1
3	INPUT	3- 1
	3.1 Basics	3- 1
	3.2 General Field	3- 1
	3.3 Cavities Field	3- 3
	3.4 PhaseLoop Field	3- 6
	3.5 Bunches Field	3- 6
4	PROGRAM STRUCTURE	4- 1
	4.1 Setting up the Program	4- 1
	4.2 scream.m	4- 2
	4.3 LoadInput.m	4- 5
	4.4 Prerun.m	4- 6
	4.5 acceleration.c	4- 8
	4.6 SimulateField.m	4-13
	4.7 detuning.m	4-29
	4.8 dimsum.c	4-31
	4.9 dophaseloop.m	4-33
5	EXAMPLE – FERMILAB PROTON DRIVER	5- 1
	5.1 Fermilab Proton Driver	5- 1
	5.2 S.C.R.E.A.M Input	5- 2
	5.3 PreRun Calculation	5- 8
	5.4 Full Run - One Pulse	5-10
6	APPENDIX A	6- 1
7	APPENDIX B	6- 9
8	APPENDIX C	6-12

1 SUMMARY

S.C.R.E.A.M is a computer code used for the simulation of the longitudinal beam dynamics in a linac composed of superconducting cavity resonators. It was developed by M. Huening (DESY) and incorporates the operating experience of the Tesla Test Facility (TTF) at DESY. It is now being adapted for the simulation of Fermilab's Proton-Driver (PD).

Optimizing the acceleration of a beam through a string of accelerating cavities requires careful control of the phases and amplitudes of the RF fields in the cavities. This is especially true in low beta ($\beta<1$) sections of the linac, where the beam velocity changes continuously, while the cavity-β changes only in a few discrete steps. The RF control system is further complicated in the case in which multiple cavities are driven by one klystron. The changing beam velocity also results in increased beam loading along the cavity string, demanding additional adjustment of the cavity fields. In addition, superconducting cavities have a very narrow bandwidth (1 kHz or less) and are therefore susceptible to minute deformations of their shape caused by microphonics or electromagnetic pressure ("Lorentz-force detuning"). These and other effects have to be taken into account in the design of the RF control system.

The S.C.R.E.A.M program is a tool to calculate the optimal phase and amplitude settings for the superconducting RF cavities in a linac. It implements fast RF control techniques such as vector-sum regulation feedback and fast ferrite vector-modulators as proposed for the Fermilab PD. The program output consists of the beam footprint in longitudinal phase-space along the linac. The program also allows exploration of the energy (E) and time (t) acceptance of the linac with the simulation of the effect of E/t jitter of the incoming beam. This program was already used to specify the range and dynamic response of the fast ferrite vector-modulators for the Fermilab PD.

This note describes the inner workings of the different modules composing the S.C.R.E.A.M program in its current form, which includes special adaptations to the Fermilab PD linac. The code is attached to this note. Extensive derivations were relegated to the appendix to keep the program description compact. Table 1-1 gives the nomenclature conventions for this document.

Table 1-1: S.C.R.E.A.M nomenclature.

Parameter	Symb	Comment
# of cavities	***Ncav***	
# of RF modules (index)	***Nm(nm)***	one RF module can contain several RF cavities
# of cavities per RF modules	***Ncm***	one RF module can contain several RF cavities
Quality factor	***QL***	loaded
Quality factor	***Q0***	unloaded
Cavity beta	β***c***	design value
# of cells	***Nc***	in cavity
Length of cavity cell	***Lc***	from cell center to cell center
Cavity frequency	ω***RF***	$2\pi fRF$
Cavity bandwidth	ω***12***	$=\omega RF/2/QL$, half-width, half-maximum
Cavity gap factor	λ	ratio of *Lc* to RF wave-length (c/f_{RF}), usually 0.5
Cavity norm shunt impedance	***r***	*Rsh/Q0*, for fundamental (accelerating) mode
Cavity loaded impedance	***RL***	*RL=rQL*, mostly *Rext* in over-coupled case
Linac coordinate	***z***	position of cavity (m)
On axis electric field	***E(z)***	(MV/m)
On axis electric field	$\hat{E}$	peak-field (MV/m)
Voltage per cavity (total)	***Vcav***	$\int E(z)dz$, real, set-field
Voltage per cavity (total)	$\hat{V}$	complex, actual field including phase factor
Relative voltage	***v***	norm. to 1, used for filling curve, to be multiplied with *Vcav*
Beam-loading voltage	Δ***Vb***	voltage reduction in cavity due to beam passage
Forward voltage	***Vfwd***	accel. voltage delivered to cavity from klystron (MV)
Forward voltage signal	***SVfwd***	voltage signal provided by control system (MV)
Forward power	***Pfwd***	Power delivered into cavity from klystron (W)
Feed forward power	***Sfwd0***	set-table value provided to control system ($MV/(\Omega)^{0.5}$)
Lorentz-detuning constant	***KL***	in $Hz/(MV)^2$, note the unusual units
Lorentz-detuning constant	***KL0***	in $Hz/(MV)^2$, initial, as defined in input
Lorentz-detuning spread	Δ***kL***	σ of Lorentz detuning const. distribution (relative to *KL*)
Lorentz-force-detuning	$\Delta\omega$***LF***	$2\pi x$ (detuning frequency in Hz)
LF-detuning compensation	$\Delta\omega$***L***	feed-forward setting to compensate for Lorentz-detuning
Microphonics-detuning const.	***Km***	microphonics detuning const. (in Hz)
Microphonics-detuning spread	σ***m***	σ of microphonics detuning const. distribution (in Hz)
Fast microphonics spread	σ***mf***	σ of fast microphonics detuning const. distribution (Hz)
Total detuning	$\Delta\omega$	Lorentzforce and microphonics detuning (rad-Hz)
Time constant	τ***c***	cavity (mechanical) time constant for detuning ("ringing")
# of time steps	***Ns***	*Ns=Nfill+Nbeam*
Time step	Δ***t***	usually 1 μsec (= bunch spacing)
Beam-on time	***tb***	=pulse time – fill time (sec)
# of time steps with beam	***Nb***	=number of bunches
Cavity fill time-constant	τ***f***	fill-time constant ($=2QL/\omega RF=1/\omega 12$)
Cavity fill time-constant	***FTayl***	fiddle factor to set the non-linearity of filling ($\in$(0.1))
Fill-time	***tf***	=pulse time – beam-on time (sec)
# of steps during fill-time	***Nf***	
Filling delay	***tfo***	cavity filling delay with respect to other cavities
# of particles per bunch	***N***	related to average pulse current with $Ib\Delta t/q$
# of macro-particles	***Nmpb***	per bunch
Particle number distribution	***n***	normalized fraction of particles contained in each macropart
Charge of single particle	***q***	usually in units of elementary charge
Mass of single particle	***m***	in units of MeV
Beam current	***Ib***	general, bunch charge / bunch spacing (A)

Macroparticle current	***Iini***	start value (A)
Macroparticle current	***I0***	start value, before fluctuations (A)
Beam energy	***E***	general particle energy (usually in MeV)
Beam energy	***Eini0***	start energy offset of synchronous particle (MeV)
Beam energy	***Eini***	start energy of mp (MeV)
Beam energy	***E0***	start energy of mp relative to bunch centroid (MeV)
Beam energy	***Ecoh***	coherent (fixed during pulse) bunch centroid energy offset
Beam energy	***Efluc***	incoherent (changes bunch to bunch) centroid energy offset
Beam energy	***Esync***	energy of synchronous particle (usually in MeV)
Arrival energy difference	***de***	with respect to synchronous particle (MeV)
Beam beta	β***b***	beam velocity normalized on c
Beam phase advance	ϕ***0***	usually negative (~-10° to -20° for protons) (rad)
Phase difference	$\Delta\phi$	with respect to synchronous phase (rad)
Transit time factor	***T'***	normalized to 1
Transit time factor	***Tsync'***	for synchronous particle, normalized to 1
Time	***t***	general (usually in sec)
Start time	***tini0***	start time offset of synchronous particle (sec)
Start time	***tini***	start time of macroparticle (sec)
Start time	***tcoh***	coherent (fixed during pulse) bunch centroid time offset
Start time	***tfluc***	incoherent (fixed during pulse) bunch centroid time offset
Arrival time	***ta***	In each cavity center (sec)
Arrival time (synchronous)	***tsync***	for synchronous particle, in center of cavity (usually in sec)
Arrival time difference	***dt***	with respect to synchronous particle (sec)
Injection time sigma	σ***t0***	for Gaussian macroparticle distribution in bunch (sec)
Injection energy sigma	σ***E0***	for Gaussian macroparticle distribution in bunch (MeV)
Injection time sigma	σ***tcoh***	coherent (same for whole pulse) (MeV)
Injection time sigma	σ***tfluc***	incoherent (different from bunch to bunch) (MeV)
Injection energy sigma	σ***Ecoh***	coherent (same for whole pulse) (MeV)
Injection energy sigma	σ***Efluc***	incoherent (different from bunch to bunch) (MeV)
Injection current sigma	σ***Icoh***	coherent (same for whole pulse) (MeV)
Injection current sigma	σ***Ifluc***	incoherent (different from bunch to bunch) (MeV)
# of branches	***Nbranch***	in macro-particle phase-space
# of sigmas	***N***σ	in macro-particle phase-space
Random number	***R***	usually obtained with MATLAB randn function
# of phase-shifters	***Ng***	usually only in the low beta section
Feedback gain	***G***	gain in vector-sum feedback loop (prop)
Forward power attenuation	***a***	attenuation of klystron forward power (relative to 1)
Vector-modulator correction	Δ***Vps***	Complex voltage, to be provided by phase-shifter (MV,rad)
Phase shift	ψ***1***	phase-shifter branch 1 (rad)
Phase shift	ψ***2***	phase-shifter branch 2 (rad)
Phase shift	ϕ***ps***	total phase shift produced by phase-shifter (rad)
Vector-modulator atten./shift	***Aps***	complex signal from PS (atten rel. to 1), phase (rad)
Vector-modul. phase offset	ψ***0***	operating point of phase-shifter (rad)
Vector-modul. reactive power	***Rps***	reflection produced by phase-shifter (relative to 1)
Phase shifter time constant	τ***ps***	dynamic response of phase shifter (sec)
Phaseshifter proportional gain	***GAps***	amplitude proportional gain (prop)
Phase shifter differential gain	***DAps***	amplitude differential gain (prop)
Phaseshifter proportional gain	***Gpps***	phase proportional gain (prop)
Phase shifter differential gain	***Dpps***	phase differential gain (prop)
Phase shifter saturation	ψ***sat***	max phase response of phase shifter (rad)
Phase-shifter aux. parameter	***Sps***	auxiliary parameter (sin(ψ ***0***)
Vector-mod reflection gain	***GR***	proportional gain of reflected signal (prop)
# of runs	***Nruns***	= # of RF pulses with a new set of *Km* for each cavity

# of files	***Nfiles***	=# of runs with a new set of *KL* for each cavity
Auxiliary parameter	α	$\beta c/\beta b$

A recent publication is recommended as a fast introduction to this program[2].

Common abreviations:

mp … macro-particle,
b …. Bunch,
m … module,
L … Lorentz,
0 … bunch-centroid,
ini … initial,
sync … synchronous,
coh … coherent,
fluc … incoherent,
cav …. cavity,
f .. fill,
fo .. fill-off,
RF … radio-frequency

Indices:

i….time steps (or bunches)
j….cavity number
l….module number
k…macro-particle
g…phase-shifter number

Other important conventions:

bunch number=time step
run=RF pulse, file=series of runs (with different KL_j)
matrix convention (ROWS, COLUMNS), as in MATLAB
array counting usually starts at 1
time step cannot easily be changed from 1 micro-sec (hard-coded)!

[2] M. Huening et al., "Simulation of RF Control of a Superconducting Linac for Relativistic Particles", Proceedings of the European Particle Accelerator Conference, EPAC 2004, July 2004 Lucerne, Switzerland

2 INTRODUCTION

S.C.R.E.A.M is a MATLAB based program that simulates the passage of a bunched beam through a linear accelerator made of superconducting cavities. It is particularly optimized for the simulation of the effects of cavity detuning due to Lorentz-force and microphonics as well as beam loading together with the corrective actions taken by the RF power system such as vectorsum control and individual cavity vector-modulators. It also takes into account jitter of the injected beam in energy, arrival-time and current. The major output parameters are listed in Table 4-3. The most important are the energy and arrival time of each particle in each cavity (and in particular in the last cavity). This information can be used to derive the beam footprint in longitudinal phase-space at any position in the linac. This is also relevant for the estimation of beam loss. Another important output consists of the operational parameters of all cavities in the linac as function of time. These include the field amplitudes and phases, the forward power and vectorsum control signal, the phase-shift and attenuation provided by the fast vector-modulators.

Each run in S.C.R.E.A.M calculates the acceleration of one beam pulse. Several runs can be simulated in one “file”. The beam pulse (or run) contains many bunches. The beam pulse is only one part of the RF pulse, which also includes the cavity filling. Obviously the beam is launched only once the filling is completed. To limit computing time the particles in a bunch are regrouped into macro-particles, which have the mass and charge proportional to the sum of the particles contained in them but are regarded as single particles during the simulation of the acceleration in the linac. The exact number of single particles contained in each macro-particle is taken into account when the beam loading is calculated. The macro-particles are distributed in longitudinal phase-space such as to simulate the distribution of particles in a real bunch. Although more granular, the bunches in S.C.R.E.A.M occupy more or less the same footprint in phase-space as the real bunches. The bunch-charge, bunch-centroid energy and arrival time at injection are randomly varied bunch-to-bunch (“incoherent”) and pulse-to-pulse (“coherent”) to explore the longitudinal acceptance of the linac. The program also provides the option to simulate different particle species.

Once the structure of the linac (= sequence of superconducting cavities) is defined in the input file, the program sequentially calculates the energy/time profile of the macro-particles along the linac. This requires the simulation of the effective accelerating voltage

provided by each cavity to each macro-particle. The effective accelerating voltage of each cavity is not only determined by the particular cavity design field, but also by the phase difference of the particle with respect to the cavity RF phase. Furthermore voltage attenuation and phase error ensue in the case of detuning of the cavity, i.e. the shift of the cavity resonance frequency from the (fixed) klystron frequency as a result of minute mechanical deformations related to Lorentz-force and micro-phonics.

Most important is the effect of the mismatch between the cavity design beta and the particle velocity. This effect is described by the transit time factor. The transit-time-factor describes the reduction of the effective acceleration of a particle due to the beta-mismatch between the cavity (βc) and the particle (βb), which is especially important in long multi-cell cavities. It also includes the effect of the sinusoidal variations of the accelerating field in the cavity in time and space on the effective acceleration of the beam. It assumes, however, that the particle is synchronous, i.e. that its phase is optimal with respect to the RF phase.

Then, typically the beam phase in proton linacs is offset by ~-20° (the so called phase advance) to obtain gradient (or phase-) focusing of the bunch. Gradient focusing consists of accelerating the beam ahead of the RF crest so that the slower particles are accelerated more than the faster particles. The gradient focusing is more effective the heavier the particles and the lower their speed. Finally, the particle phase difference due to injection jitter scatters the particle phase around the phase advance setting. The phase-factor is also strongly affected by detuning and beam loading in the cavities because of the ensuing variations of the amplitude and phase of the cavity accelerating voltage and thus on the cumulative acceleration history of each particle.

Once the corresponding beam phases are calculated, the induced field (or beam loading) is subtracted from the cavity field. The re-fill of the cavity is simulated assuming constant klystron power. The forward power from the klystron to the cavities is adjusted (with feed-forward and feedback) such as to keep the cavity voltage and phase constant. The feed-forward settings take into account the expected beam loading and Lorentz-force detuning. Since RF-modules typically consist of several cavities, special techniques are needed to regulate the RF power and phase in the individual cavities, which may differ in their detuning and beam-loading. At the klystron level the phase and amplitude of the RF signal is typically regulated using vector-sum

regulation. Vector-sum regulation consists in summing the vectors, defined by the phase (direction of vector) and amplitude (length of vector) measured in each cavity, to derive the klystron RF phase and amplitude setting that produce the desired sum vector. In the Fermilab PD it is also proposed that the phase and amplitude in the individual cavity be regulated with fast ferrite vector-modulators (or E-H tuners). S.C.R.E.A.M is capable of simulating both vector-sum control at the module level and individual cavity vector-modulators.

Constant phase offset settings such as the synchronous phase and the beam phase advance are programmed into the low level RF system that drives the klystron RF phases with respect to the master oscillator (and with wave-guide length differences as well as the three-stub tuners for the cavity-to-cavity differences within a module). The program obviously assumes that the synchronous phase was determined and all phases used are therefore with respect to this phase.

With a typical time-step of 1μs, a thousand bunches are simulated for a typical beam-pulse of 1ms duration. A bunch moving with *c* travels 300 m during 1μs. A bunch is therefore usually at the end of the linac before the next is launched. The program therefore describes the progression of a bunch through the entire linac within one time-step[3]. Obviously the many macro-particles of which the bunch consists are also tracked independently through the linac. The cavity field calculations (as well as the update of the feedback loops, etc..) are performed once per time-step, hence the linac field configuration is essentially static during the passage of the bunch. The previous bunches affect the actual bunch only through the accumulated beam-loading voltages.

The program produces a huge amount of data. All the data for the first pulse simulated are saved in *preresults.mat*. For all other *Nfiles* times *Nruns* pulses, only a subset of the data is stored. Regarding the beam properties, for instance, the program only keeps the final energy and phase error of the macro-particles in each bunch. Regarding the cavity properties all relevant parameters for each cavity are stored, with a point every *Downsample* time-steps. The data matrices are stored in structures, while the matrices for all *Nfiles* times *Nruns* pulses are stored in structure arrays. The so-called *randstate* variable

2 In the Fermilab Proton Driver the bunch spacing is much smaller that 1 μsec. The input into SCREAM takes this into account by grouping as many PD bunches into a super-bunch such as to keep the average beam current to the design value. This approach is valid for the calculation of beam-loading, for example, but would need revision if space-charge effects are to be considered.

is also stored for each pulse (run), such that the pseudo-random number series needed to define the beam injection jitter, Lorentz-detuning and microphonics can be recalled for detailed analysis. The data arrays are saved in *cavresults.mat* and *beamresults.mat* MATLAB data files.

No wake-field effects, space-charge effects as well as intra-beam scattering effects between macro-particles or bunches are considered. Future additions to the program could include transverse focusing (first experiments with a TRACE3D interface were already made) in order to calculate the transverse phase space. The simulation of space charge effects would certainly be another important contribution.

The program operates in a MATLAB environment. MATLAB is a special purpose computer program, optimized to perform scientific calculations. It implements MATLAB language and provides a very extensive library of pre-defined functions. It is in particular optimized for the handling of matrix mathematics. That simplifies the handling and plotting of the large amounts of data calculated with a beam-tracking program of the kind discussed here. The main disadvantage of MATLAB is that it is slower than compiled languages, the consequence of being an interpreted language. For that reason S.C.R.E.A.M uses C-language programs in the tracking part of the program. The c-routines, however, are slowed down by the MATLAB-C data format conversion steps required. S.C.R.E.A.M consists of a lose assembly of several files, which need to be installed together. S.C.R.E.A.M should be launched from the directory where all the Scream files are located. These files are listed in Table 4-1. The input file, linac.csv, is not listed there. In order to tell the program where to look for the input file, the *datadirectory* variable needs to be set. Per default *datadirectory='run0data'*, thus it is best to copy linac.csv into the folder *run0data*, which needs to be in the same directory as the Scream files. The program is started by typing s*cream* into the MATLAB command window, which starts the **scream.m** script. Before calling **scream.m**, however, the *debugging* variable, which discriminates between the different implementations of the program, can be specified. If debugging is not set, the default value of zero is assumed and the complete program version is launched. Other debugging options are discussed in chapter 4. The input file needs to be in the form of a comma-separated list file, which uses MATLAB nomenclature to define structure variables ({ }), comments (%), etc. The solution chosen here is to save the Excel spreadsheet as a .csv type file. The program can be run both in a Unix and Windows environment, once the respective versions of MATLAB are installed.

3 INPUT

3.1 Basics

The input into S.C.R.E.A.M consists of defining several MATLAB structure variables. MATLAB structure variables can contain multiple arrays of arbitrary dimensions. In this case the structures contain mostly vectors and matrices. The units are usually S.I. with the only notable exception that accelerating voltages in the cavities are assumed to be in MV and that beam and particle energies are given in MeV. That allows the accumulated accelerating voltage to be directly translated into beam energy. To cut processing time, as many calculations as possible are performed in the input, generated with an Excel spreadsheet. The input file is saved as a comma-separated list file (csv) in the *run0data* directory to be read by MATLAB routines. Note that MATLAB interprets the '%' as comment delimiter. The structure variables defined in the input are called *General*, *Cavities*, *Phaseloop* and *Bunches*. *General* defines general variables such as step-size and the number of output files. *Cavities* defines the properties of all cavities in the linac in the order in which they are installed. *Phaseloop* contains the parameters of the individual cavity phase shifters, if there are any. *Bunches* defines the beam properties, such as macro-particle charge and phase-space distribution at the start of the linac.

Each beam-pulse contains as many bunches as there are time-steps during the beam-on part of the RF pulse (which also includes the filling time). The total number of pulses simulated is given by *Nfiles* x *Nruns*. The separation into two groups of files allows the independent variation of different parameters. For instance the set of Lorentz-force detuning constants distributed randomly over all cavities is defined once for all *Nruns* within one *Nfiles*, while the slow/fast micro-phonics constants of all cavities are randomly recalculated for every *Nruns*/*Ns*.

The input-file, **linac.xls**, allows initialization and definition of the structure variables introduced above. These variables are discussed in detail in the following.

3.2 General Field

The *General* structure variable consists of eleven scalar fields, which hold a set of general parameters as listed in Table 3-1. Most of them

are parameters describing the jitter of the beam energy, arrival-time and current at injection. *General* provides the half-width sigmas of the Gaussian distributions of the bunch centroid (or bunch mean) energy (in MeV), starting time (in sec) and current (in %) for each pulse (coherent) and for each bunch (incoherent). (Note that, as will be discussed in further detail in section 3.5, the particles within the bunch are also Gauss-distributed in longitudinal phase-space around the centroid.) The half-widths of the Gaussian particle number distributions within the bunches are given in the *Bunches* field. The sigmas are multiplied with a MATLAB generated random number ($\in$(-1,1)) in subsequent sections of the program to derive the actual bunch centroid starting energy, time and charge. To obtain statistical significance several runs (= beam pulses) need to be performed for each set of input parameters.

A bunch is launched into the linac at every time step during the duration of the beam pulse. In a further extension of this principle there are in fact two parameters: *Nfiles* – the number of files - and *Nruns* - the number of RF pulses per file. The separation into *Nfiles* and *Nruns* allows for the variation of different parameters from one set of runs to the next. Currently, for instance, the Lorentz-force detuning constant is randomly varied from file to file. The microphonics detuning on the other hand, is randomly varied from run(pulse)-to-run (slow) or bunch-to-bunch (fast).

The General structure also contains the fast ferrite vector-modulator dynamic response time constant, since this number is subject to change depending on the ongoing hardware development.

Table 3-1: Fields in the *General* structure-variable. *added by **LoadInput.m** during runtime.

Field	Comment	typical
Title	string	-
Efluc	σ of distrib. of bunch centroid initial *E* (incoh.) (*scal*)	50 keV
Tfluc	σ of distrib. of bunch centroid start time (incoh.) (*scal*)	5.8 ps
Ifluc	σ of distrib. of total bunch charge (incoh.) (*scal*)	1%
Ecoherent	σ of distrib. of bunch centroid initial *E* (coh.) (*scal*)	50 keV
Tcoherent	σ of distrib. of bunch centroid start time (coh.) (*scal*)	5.8 ps
Icoherent	σ of distrib. of total bunch charge (coh.) (*scal*)	1%
Stepsize	time step of calculation, bunch spacing (*scal*)	1 μs
Downsample	reduction factor for program output time-step (*scal*)	1
Filltime	fill-time of cavities (applies to all cavities) (*scal*)	500 μs
Beamtime	duration of beam pulse (*scal*)	800 μs
PhaseTau	phase-shifter dynamic response time (*scal*)	150 μs
Nfiles	number of simulations (*scal*)	1
Nruns	number of runs per simulation (*scal*)	1
doPhaseloop	1 or 0, depending on whether *Phaseloop* exists or not	1*

3.3 Cavities Field

The cavity parameters needed in the simulation are listed in the *Cavities* structure of the input file. The *Cavities* fields are listed in

Table 3-2: Fields in the *Cavities* structure-variable. All fields are of dimension (*Ncav*). *overwritten during runtime by Prerun.m.

Field	Comment	unit	symb
CavNo	number of cavity	-	***j***
Module	number of Module to which the cavity belongs	-	***nm***
Beta	design-beta of the cavity	-	β***c***
Position	position of the cavity center along linac (calcul. from n-1 position and *Neighbour*)	m	***z***
Neighbour	distance between cavity center and center of previous cavity	m	Δ***z***
AmpGen	$\int E(z)dz$ as obtained from cavity design codes	MV	***Vcav***
Amplitude	set-amplitude used in calculation, can be different from Ampgen	MV	***Vcav***
Phase	beam phase advance (relative to synchronous phase, usually negative, set manually)	deg	ϕ***0***
Feedback	proportional feedback gain for module, will be overwritten during runtime in LoadInput.m with average gain/cavity	prop.	***G***
Cells	number of cells in cavity	-	***Nc***
GapLambda	ratio of cavity cell length / RF wave-length (typically 0.5)	-	λ
Time	arrival time of particle in cavity, automatically calculated by program when (–1), will be overwritten by Prerun.m with synchronous phase	s	***tsync***
Frequency	cavity operational frequency	Hz	***fRF***
QloadedAve	external cavity Q for fixed input coupler setting	-	***QL***
Qloaded	actual value used in calculation, can be different from QloadedAve	-	***QL***
Attenuation	attenuation of forward signal, relative to forward amplitude	rel.	***a***
Rshunt	normalized cavity shunt impedance, *Rsh/Q0*	Ω	***r***
Microphonics	σ of cavity frequency shift (Gaussian) due to slow microphonics	Hz	σ***m***
FastMicroph.	σ of cavity frequency shift (Gaussian) due to fast microphonics	Hz	σ***mf***
Mode	e.g. 3.1415 for π-mode	(rad)	–
dw	feed-forward detuning of cavity to anticipate Lorentzforce detuning, = 2π*xKL0xVcav*^{2} , will be overwritten during runtime in scream.m using the actual KL including the random variation for every *Nfiles*	(rad-Hz)	$\Delta\omega$***L***
KLorentz	Lorentzforce detuning constant, note the unusual units (integation over cavity length (division by (*LcxNc*)^2) is done in the input)	Hz/(MV)^2	***KL0***
Kspread	σ of Gaussian distribution of Lorentz-force detuning constants, relative to KLorentz	(0,1)	Δ***KL***
FillOff	delay of filling start (to allow for faster filling cavities, so that all cavities are filled at the end of the fill-time)	μs	***tfo***
BeamEnergy	energy of synchronous particle (=*q·Vcav·T''*), will be overwritten by Prerun.m with E of synchronous particle incl. beam phase advance	MeV	***Esync***
BeamBeta	beam velocity, relative to *c*, calculated from BeamEnergy	-	β***b***
TTF	transit time factor, will be overwritten by Prerun.m	-	***T'***
Atten	not used (historical), added in LoadInput.m (default=*1*)	rel.	-
FillTau	cavity fill-time, added in LoadInput.m if not specified (=*2QL/*ω*RF*)	sec	τ***f***
FillTaylor	fill-curve shape factor, will be added in LoadInput.m ($\in$0,1)	-	***FTayl***
ReactiveAmp	gain of phase shifter reflected amplitude - added in LoadInput.m (=*0*)	-	***prop***
Reactive	gain of phase shifter reflected phase - added in LoadInput.m (=*0*)	-	***prop***
Egain	energy gain of sync. part. in each cavity, will be added in Prerun.m	MeV	***dE***

Table 3-2.

The *Cavities* structure essentially contains all the relevant information to describe the linac hardware (except for the fast vector-modulators). In particular it encodes the cavity sequence. It consists of 27 vectors with *Ncav* elements, where *Ncav* is the number of cavities in the linac. The most important parameters are obviously the position of each (center of the) cavity, its length, its number of cells, the identifier of the RF module to which it belongs, the design frequency and beta, the shunt impedance and loaded Q and the average accelerating voltage over the cavity at design amplitude as well as its detuning characteristics (fast and slow micro-phonics and Lorentz-force detuning parameters). The issues related to detuning are discussed in detail in chapters 4.2, 4.6 and 4.8. The *Cavities* structure also includes settings, which are part of the amplitude and phase control of the cavities. Among them for example the fill delay and the (vector-sum) feedback gain (*G)* and individual cavity directional coupler attenuation (*a)*. Finally it also includes some information about beam settings, such as the beam phase advance in each cavity, $\phi 0$, for phase focusing.

The so-called *Ampgen* parameter is the total accelerating voltage, integrated over the length of the cavity as given in Eq. (3-1) for the case of a sinusoidal field shape. $\hat{E}$ is the on-axis peak electric field in (MV/m), *Lc* the length of a cell and *Nc* the number of cells per cavity. Note that Eq. (3-1) is only an approximation, since it calculates the effect of a multi-cell resonator by multiplying the voltage provided in one cell by the number of cells *Nc*. A more accurate calculation can be performed with a finite-element code that computes the exact field distribution, including drift sections between cells and the reduced fields in the end-cells. It also does not apply to cases with non-sinusoidal spatial field profiles (e.g. pill-box cavities).

$$Vcav \approx Nc \int_0^{Lc} E(z)dz \approx Nc \int_0^{Lc} \hat{E} \cos\left(\left(\frac{z}{Lc} - \frac{1}{2}\right)\pi\right)dz \quad (MV) \tag{3-1}$$

The amplitude (*Amplitude*) that is actually used as the target acceleration voltage in the simulation can be different from the design amplitude. This can be useful to counteract varying transit time factors across a module (by increasing the field where the transit time factor is small, for example).

The *Phase* factor allows defining a phase advance of the individual cavities. In the particular case of S.C.R.E.A.M it mostly serves to set the beam phase advance in each cavity. The phase-advance is important for phase focusing, especially at low βb. Note that, as a result of the way the model calculates the phase delay of the particles in each cavity, the small phase offset due to the finite travel-time from cavity to cavity does not need to be included in the phase-offset for each cavity provided in the input file. S.C.R.E.A.M automatically computes the synchronous phase (see the description of the **Prerun.m** routine).

The proportional feedback (*Feedback*) is used for the vector-sum control feedback. Although the vector has *Ncav* elements the values in the input file are the gains applied to the sum signal in each module. This vector is recalculated in **LoadInput.m** to divide the total gain in each module by the number of cavities in each module. After the transformation the Feedback vector contains the average gain per cavity. This implementation was chosen because the vector-sum gain in the low level RF system is of course the gain for the entire module (and this is what needs to be given in the input). The feedback gain is chosen higher in modules belonging to the low-beta section, because it makes the feedback loop more sensitive and gives faster time response. The feedback gain is somehow relaxed in the $\beta=1$ section. Note that cable effects, temperature variations in the electronics and noise in the detectors are also amplified with this gain! Also defined is the feed-forward power attenuation *a* (relative to 1) to describe the possible attenuation of the directional couplers routing the power to the individual cavities. This factors is actually used to set small cavity-to-cavity variations of the forward power as needed to reduce the effects of beam loading variations along the cavities in a module. The TESLA experience has shown that $a_j \sim T'^{-0.25}$ (where *T'* is the normalized transit time factor) gives good results.

The *Time* parameter allows pre-setting of the arrival time (e.g. in case delay-loops or damping rings are to be included in the lattice). Typically, however, it is set to (-1), which tells the program that the arrival time is to be calculated by the program. The arrival time calculation is a major part of the **acceleration.c** routine. Using this routine the **Prerun.m** script calculates the arrival time and energy gain of the synchronous particle in each cavity.

The cavity loaded Q (*QL*) is typically set to a value derived from the expected beam loading. The *QL* actually used in the program can differ slightly from the actual loaded Q. In driving the cavity off the optimum

coupling condition (and raising the forward power by a similar factor), the differences in beam-loading between the cavities in a module can be leveled.

The input sheet also calculates the transit-time factor, *T'*, and the beam energy (beam-beta, βb) of the synchronous particle in each cavity. This calculation also serves to derive optimized settings of the cavity voltage and attenuation as discussed above. The transit time factor is discussed in further detail in chapter 4.5 in the context of the description of the **acceleration.c** routine.

The *fill-off* parameter allows delaying of the filling of faster filling cavities, such that all cavities are ready to accept beam at the same time. The slowest filling cavities are filled according to the fill-time defined in *General* and have a *fill-off* of zero. This feature is especially important to take into account the change in filling time as a result of tuning the *QL* of the cavities off optimum. The additional parameters *FillTau* (cavity fill time constant, $=2QL/\omega RF$) and *FillTaylor* (determines deviation of fill-curve from linear shape, with 1 giving the linear shape) are also used to describe the cavity filling. The *FillOff, FillTau and FillTaylor* vectors are written by the **LoadInput.m** routine, if they are not specified in the input file. The same is true for Atten, *ReactiveAmp, Reactive* and *Egain*.

Six additional vectors are added by the program (*Atten, FillTau, FillTaylor, ReactiveAmp, Reactive, Egain*). *ReactiveAmp* and *Reactive* describe the gain of the signal reflected by the fast vector-modulator.

3.4 Phase Loop Field

The *Phaseloop* structure contains information about the fast ferrite vector-modulators, a particular hardware component added to some of the PD cavities. The fast vector-modulator allows to regulate phase and amplitude of the individual cavities in order to provide the additional regulation that the vector-sum control loop cannot provide. It contains eleven fields, of which only *CavNo* and the major proportional and differential gain factors (for the PID regulator) are actually used. All other parameters are currently “hardcoded” and cannot be varied individually for the cavities, as it would be the case in reality. Table 3-3 summarizes the *Phaseloop* parameters. *CavNo* gives the identifier of the cavities, which have a phase-shifter. *Gain* and *DGain* are the proportional and differential gains for the phase-signal

in the PID type vector-modulator implementation. AmpGain and AmpDGain are the equivalent parameters for the amplitude signal.

The currently used input file also includes fields that can be useful for future implementations of the program, but are not used at this point.

Table 3-3: Fields in the *Phase-Loop* structure-variable. All fields are of dimension (*Ng*).

Field	Comment	unit	symb
CavNo	number of Cavity (renamed to *PLIdx* later)	-	***j***
Gain	proportional gain for phase correction signal	prop	***Gpps***
AmpGain	proportional gain for amplitude correction signal	prop	***GAps***
DGain	differential gain for phase correction signal	prop	***Dpps***
AmpDGain	differential gain for amplitude correction signal	prop	***DAps***
Atten0	placeholder-not yet used,	-	**-**
PhIni	placeholder-not yet used,	-	**-**
KPSnb	placeholder-not yet used,	-	**-**
KSlnb	placeholder-not yet used,	-	**-**
KBFnb	placeholder-not yet used,	-	**-**
KCFnb	placeholder-not yet used,	-	**-**

3.5 Bunches Field

The *Bunches* field defines the composition (number, charge and mass) and the initial phase-space distribution (starting times and starting energies as well as current) of the macro-particles in a bunch. The mass (in MeV) and charge (in units of elementary charge) of the macro-particles is set to the same as that of the single particles, since the calculation of the effective acceleration in the cavity just depends on the charge to mass ratio. The number of single particles contained in the macro-particles, however, is important for the calculation of the beam-loading in the cavities. The macro-particle populations obviously need to add up to the specified total bunch population, which in turn needs to be consistent with the average pulse current of the to be simulated beam.

The general phase space distribution of the macro-particles as defined in *Bunches* in its current implementation is shown in Fig. 3-1. The *Nmpb* macro-particles per bunch are distributed in a “star-pattern” along *Nbranch* lines azimuthally spread with the *Angle* parameter over 2π (*360°/Nbranch* angles). The particles are distributed at *Nbranch* (typically 8) discrete angles in the starting time / starting energy plane at up to $N\sigma$ (typically 9) energy- and time-sigmas, $\sigma E0$ and $\sigma t0$. Therefore the *Bunches* vectors typically have ($Nbranch \times N\sigma)+1$

elements. Eq. (3-2) defines the functions with which the phase space coordinates of the macro-particles are calculated. The dashed lines in the figure indicate contours of constant sigma. The contours in Fig. 3-1 were calculated for a *Sigma Step* of 1 σ. The center point in Fig. 3-1 corresponds to the synchronous particle, which arrives at time zero at the center of the first cavity. This distribution centroid, however, varies from case-to-case as discussed in the context of the *General* input field and as will be discussed further below. The header parameters of the *Bunches* structure, *Input Time* (sec) and *Input Energy* (MeV), are the coordinates of the synchronous center-point of the particle distribution in phase-space (*tini0*, *Eini0*). These parameters are essentially the constant offsets of the phase-space distribution centroids to which the variations defined in the *General* input field will be added later in the program. The *Time Sigma* (sec) and *Energy Sigma* (MeV) given in *Bunches* are the half-widths of the Gaussian macro-particle distributions in phase space, which are invariable in the program (while the centroid is subject to change). The *Sigma Step* describes the number of sigmas, which are to be put between two macro-particles and thus describes the coarseness of the macro-particle distribution in phase-space. Note that the phase-space in Fig. 3-1 is given in the most general form, namely in terms of sigmas of the energy/arrival-time distributions. The sigmas are then quantified in *MeV* and *secs* in the header of the *Bunches* structure. Fig. 3-1 also indicates the order in which the macro-particles are defined in *Bunches* (arrows).

The number of particles contained in each macro-particle decreases according to a Gaussian function such as shown in Fig. 3-2. The macro-particle population is calculated with Eq. (3-3). The macro-particle current is also derived from the particle number distribution (Eq.(3-4)). The input spreadsheet performs the calculation of the Gaussian particle number distribution of the macro-particles in phase space from the Gaussian mean and width, given in the header of the *Bunches* field with this formalism. This distribution is normalized to one and needs to be multiplied with the total number of particles per bunch N to obtain the number of particles per macro-particle. The total number of particles/bunch is adjusted such as to give the average pulse current specified for the to be simulated beam.

Table 3-4: Fields in the *Bunches* structure-variable. The fields in the rows starting with "Angle" and below have dimension *Nmpb*. All other fields are scalars.

Field	Comment	unit	symb
Input Time	time offset of mean (centroid) of Gaussian particle distribution in all bunches	sec	***tini0***
Input Energy	energy offset of mean (centroid) of Gaussian particle distribution in all bunches	MeV	***Eini0***
Time Sigma	sigma for time-dimension of Gaussian particle distribution	sec	***σt0***
Energy Sigma	sigma for energy-dimension of Gaussian particle distribution	MeV	***σE0***
Sigma Step	distance betw. macro-particles in relative phase space (multiples of σ)	-	***-***
Angle	angle in initial time / energy phase space for macro-particle (360°/Nbranch)	deg	-
Offset	number of sigmas from mean (centroid) in Gaussian particle number distribution	-	***Nσ***
Energy	initial energy of macro-particle	MeV	***Eini***
Time	arrival time of macro-particle in first cavity	sec	***tini***
Mass	rest-mass of particles (to discriminate e^- from e.g. H^-)	MeV	***MeV***
Charge	charge of particles in units of the elementary charge	-	***q***
N	number of particles per macro-particle	-	***n***
I	"current" of macro-particle (NqΔt, where Δt is the bunch spacing)	A	***I0***

$$E0_k(m,n) = Eini0 + u \cdot \sigma E0 \cdot \cos\left(v \frac{360^\circ}{Nbranch}\right) \quad u = 0,1,....., N\sigma$$
$$t0_k(m,n) = tini0 + u \cdot \sigma t0 \cdot \sin\left(v \frac{360^\circ}{Nbranch}\right) \quad v = 0,1,....., Nbranch \qquad (3\text{-}2)$$

Note that the lines in Fig. 3-1 fall in between sigmas, namely at integer multiples of $\Delta\sigma/2$. The sigmas refer to the Gaussian particle distribution that defines the population of each macro-particle in the bunch. The lines in the plot define the areas in phase-space over which the Gaussian distribution is integrated to calculate the population of each macro-particle. The relative particle number distribution, $n(\sigma)$, is given in Eq. (3-3). Note that the population of the centroid macro-particle is calculated separately. Also note that the calculations are given for a general $\Delta\sigma$, which does not have to be an integer multiple of sigma. The relative number distribution integrates to 1. The n_u needs to be multiplied with the number of particles per bunch N to calculate the total number of particles per macro-particle. Since in the real machine there often are several bunches during one time step of

S.C.R.E.A.M (1 μsec), the charge needs to be accordingly redistributed into bigger, but less frequent bunches for the purpose of the simulation. The total number of particles per bunch is calculated from the average beam current of the real machine (that is to be simulated) and the time-step as given in Eq. (3-4).

$$n_0 = \frac{1}{2\pi\sigma^2}\int_0^{2\pi} d\phi \int_0^{\Delta\sigma/2} r dr e^{-\frac{r^2}{2\sigma^2}} = 1 - e^{-\frac{\Delta\sigma^2}{8\sigma^2}} = 0.118 \qquad u = 0$$

$$n_u = \frac{1}{2\pi\sigma^2 N_{branch}}\int_0^{2\pi} d\phi \int_{\Delta\sigma\left(u-\frac{1}{2}\right)}^{\Delta\sigma\left(u+\frac{1}{2}\right)} r dr e^{-\frac{r^2}{2\sigma^2}} = \frac{1}{N_{branch}} e^{\frac{(2u-1)^2}{N_{branch}}\frac{\Delta\sigma^2}{\sigma^2}}\left(1 - e^{-\frac{u\Delta\sigma^2}{\sigma^2}}\right) \qquad u = 1,2,.....$$

$$n_{tot} = n_0 + N_{branch}\sum_{u=1}^{\infty} n_u = 1 \qquad (3\text{-}3)$$

$$I0_k = \frac{n_k \times N \times q}{\Delta t},\ Ib = \sum_{k=0}^{Nmpb-1} I0_k \quad (A) \qquad (3\text{-}4)$$

The calculation in Eq. (3-3) is performed for circles in phase space. These circles can be deformed into ellipses through assignment of a particular $\sigma E0$ and $\sigma t0$ (e.g. 60 keV and 9.9 ps in the proton driver).

The phase space jitter of the bunch centroid at injection as well as the fluctuations in bunch current are not calculated in the *Bunches* field, but in the **SimulateField.m** routine of the program on the basis of the statistical distribution widths given in the *General* field. The program uses a random number generator to shift the center-point of the distribution a fraction of sigma in starting time, starting energy and total bunch current. The pulse-to-pulse (run-to-run) variations of the time and energy centroids of the distribution as well as the bunch current are calculated with the coherent sigmas. The bunch-to-bunch variations of the time and energy centroids of the distribution as well as the bunch current are calculated with the fluctuation (incoherent) sigmas. The sum of the coherent and incoherent variations are then added to the particle energy and time offset defined in *Bunches* to produce the final phase-space centroid of the distribution used in the bream tracking calculation. The random number generator function parameters are stored with every seed such that the calculations can be repeated. More details on this procedure can be found in the discussion of **SimulateField.m**.

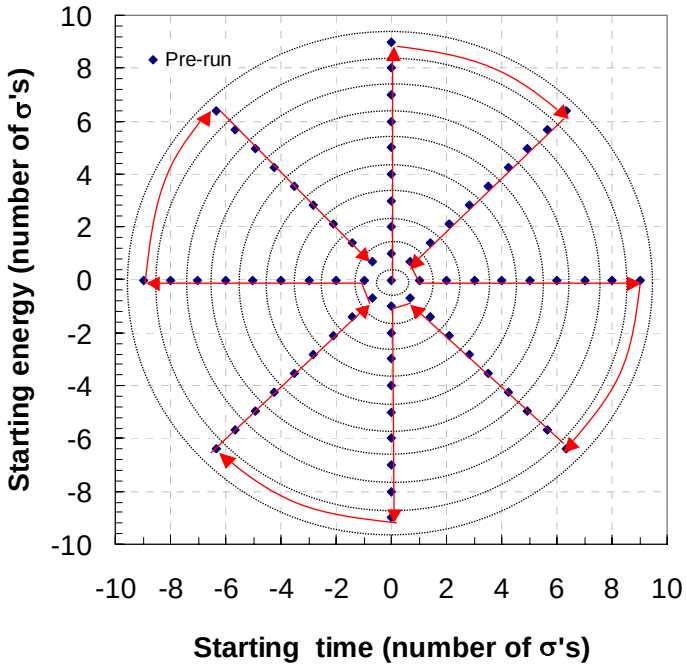


Figure 3-1: Initial distribution of macro-particles in longitudinal phase-space. The distribution is in number of sigmas of the Gaussian distributions. The rings delimit the phase-space surface area allocated for each of the n-sigma zones. Given that the representation is independent of the exact energy and time sigma these zones are rings. All bunches in the simulations use this phase-space distribution, except for coherent variations of all particles. Also shown is the order (starting with the center particle) in which the macro-particles are defined in the input file (red arrows).

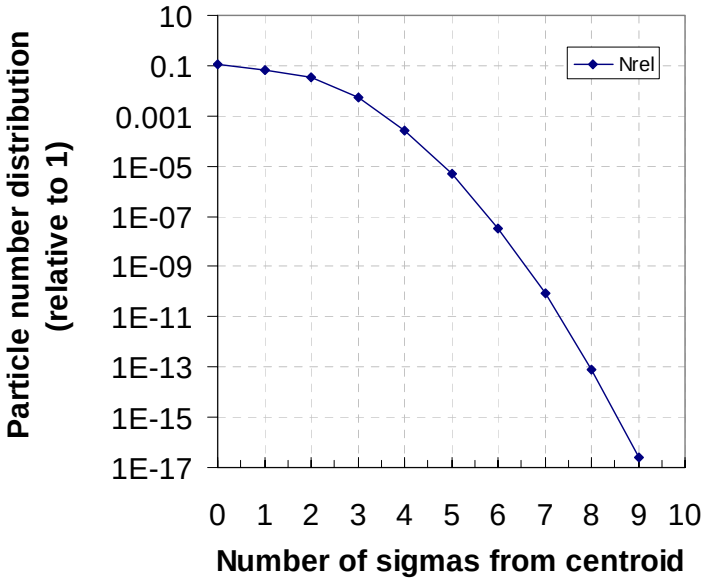


Figure 3-2: Normalized distribution of number of particles $n(\sigma)$ per macro-particle as function of number of σ from the centroid. Note that the relative particle number of the centroid is calculated separately and that the sum of all particles (including the other N_{branch}-*1* branches) is 1.

4 PROGRAM STRUCTURE

4.1 Setting up the Program

To execute the program all program-files need to be copied into the working directory. Most of them are MATLAB type m-files. The list of files is given in Table 4-1. They will be discussed in detail in the following. The program is started by calling the main routine (scream) in the MATLAB environment (typing scream in the MATLAB environment). This launches the **scream.m** script. Note that first the working directory in MATLAB needs to be set to the directory in which all files are located. Two of the files (**acceleration.c** and **dimsum.c**) are in c-code and need to be compiled once in MATLAB before the program can run. The compilation, which produces the files, can be done with the mex command from the MATLAB environment.

Before calling the **scream.m** script some program parameters need to be set, among them the *debugging* variable (default: 0) and the data-directory variable (default: *'run0data'*). The *debugging* variable gives choice between different levels of program execution (see 4.2 for more details). The directory string variable *datadirectory* contains the name of the folder, where the input file is located. The input loading routine assumes that this folder is in the working directory. The basic start-up commands are repeated below. The program will create the *cavityresults.mat* and *beamresults.mat* datafiles, which contain the results. The results of the first (debugging) run are stored in *preresults.mat*. The data can be reloaded with the MATLAB *load* command.

```
debugging=0;
datadirectory='run0data';
scream;
```

Table 4-1: Files required for the execution of Scream. Files with extension '.m' are Matlab files.

Filename	Comment
scream.m	main program, executes different versions of program, saves data
LoadInput.m	reads input file linac.csv
Prerun.m	tracks synchronous pilot particle and the "synchronous" bunch
acceleration.c	tracking routine, calculates E(t) for all macroparticles
SimulateField.m	calculates cavity fields including detuning and feedback control of amplitude and phase
detuning.m	calculates Lorentzforce detuning due to cavity voltage
dimsum.c	averages (complex) voltages of all cavities for each module for vectorsum control
initiphaseloop.m	initializes parameters for fast vector-modulator implementation
dophaseloop.m	simulates the fast vector-modulator

4.2 scream.m

Scream.m, albeit short, is in fact the main program, with as its main purpose to call other scripts depending on the value of the *debugging* variable, which discriminates between different program versions. If debugging is not defined in the MATLAB environment before the execution of **scream.m**, the variable is automatically set to zero. Table 4-2 lists the different program options launched according to the value of the debugging variable. Most of them consist in calling the **Prerun.m** routine. The **Prerun.m** routine tracks a so-called pilot particle and a pilot bunch through the linac. The pilot particle is synchronous and therefore defines the synchronous phase.

Table 4-2: Different program options that can be selected with the debugging parameter.

debugging	action	Comment
0 (default), >3	complete run	simulates *NfilesxNruns* complete RF pulses (*Nb* bunches with *Nmpb* macro-particles)
1	pre-run, beam only	simulates the synchronous pilot particle and one synchronous bunch
2	pre-run, beam & cavities	pre-run with one RF pulse, includes simulation of cavity fields (incl. feed-back & detuning)
3	pre-run, beam & cav + save	pre-run with one RF pulse, includes simulation of cavity fields, with save (preresults.mat)

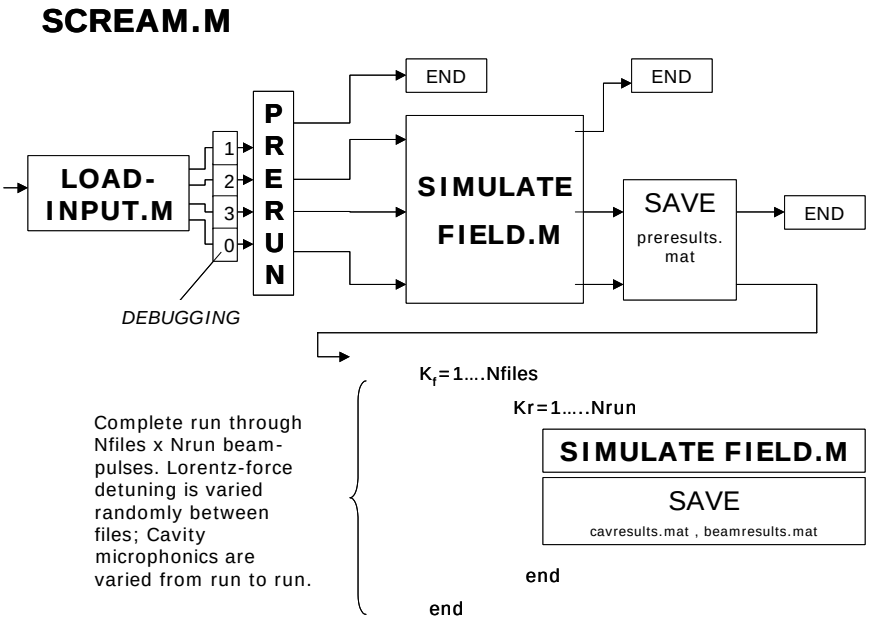


Figure 1: Flow-chart for scream.m. The Loadinput, Prerun and Simulatefield modules are discussed in further sections of this document.

Only in the case of *debugging=0* (or>3) is the program fully executed. The full execution consists of two loops over the *Nfiles* and *Nruns* RF pulses. The *Cavities* structure is redefined in this case as *SimCav* to allow for a redefinition of the *KLorentz* vector. With every cycle through the outer (*Nfiles*) loop the *KLorentz* vector is redefined with Eq. (4-1). This implementation was chosen to take into account the fact that the individual cavity Lorentz-force detuning constants should be invariable once the accelerator is built. In that sense different *Nfiles* describe different accelerators (with the same layout). Note that the program assumes that the Lorentz-detuning factor, which is usually given in *Hz/(MV/m)*^{2}) was already divided by the square of the cavity length in the input.

$$\boldsymbol{KL_{j,new} = KL_{j,old}\left(1+\Delta kL_j R_j\right)} \quad \left(\frac{\boldsymbol{Hz}}{\left(\boldsymbol{MV}\right)^2}\right) \tag{4-1-a}$$

"New" and "old" in Eq. (4-1) refer to the KL_j vector of the former cycle (or *KL0* as defined in *Cavities* in the case of the first cycle). *Rj* stands for a vector of pseudo random numbers for cavity *j* produced with the MATLAB *randn* function. These random numbers follow a (normalized) Gaussian distribution around zero with variance 1. The frequency change in each cavity due to fast (slow) microphonics is randomly varied for each bunch (pulse) in the **SimulateField.m** routine, which is called once every run to perform a simulation of an RF pulse. The **scream.m** script also redefines the Lorentz-force pre-detuning that was previously defined in the input on the basis of the *KL0* detuning constants given in the input (*Cavities.Phase*). This ensures that the Lorentz-force pre-detuning is as precise as possible.

$$\boldsymbol{\Delta\omega L_{j,new} = \Delta\omega L_{j,old} - 2\pi\left(KL_{j,new} - KL_{j,0}\right)Vcav_j^2} \quad \left(\boldsymbol{rad - Hz}\right) \tag{4-1-b}$$

The results of the simulation of an the first bunch (*debugging =2&3*), consisting of *Nmpb* macro-particles distributed over phase-space, are contained in the *cavpre* and *bmpre* structures, which contain the main parameters of the cavities and of the beam for all time steps in the pulse as listed in Table 4-3 (see discussion of **SimulateField.m** for further details). These fields can be very large and they are stored in the *preresults.mat* MATLAB data file created by **scream.m** in the *datadirectory* directory.

For a complete run the output structures are named *cr* (cavity-results) and *br* (beam-results). They are organized into the *beamresults.mat*

Table 4-3: S.C.R.E.A.M output parameters.

Parameter	Comment	units
Beam		
Time	arrival time (phase) of all macroparticles/bunch in each cavity (pre-results) or in final cavity (final results)	sec
Energy	energy (velocity) of all macroparticles/bunch in each cavity (pre-results) or in final cavity (results)	MeV
Cavities		
CField, SField	accelerating voltage in cavity incl. phase (actual & set)	MV
CForwd, SForwd	amplified control signal for vector-sum control – difference of set-voltage and actual voltage; (actual & set, incl. feed-forward & feedback)	$MV/\Omega^{0.5}$
CDrive	amplified control signal for vector-sum control including phase-shifter – (incl. feed-forward & feedback)	MV
dw	detuning of each cavity at each time step	rad
CCur	beam loading factor incl. phase (rel. to synchronous)	A
ECur	bunch final E fluctuation (rel. to synchronous)	MeV
Phase shifters		
CFwdpl	forward attenuation/phase-shift in each phase-shifter / time step (after feedback)	relative
CRvspl	reflection from each phase-shifter / time step (after feedback)	relative
sh1,sh2	phase shifter corrections for amplitude and phase	rel/rad
ps1,ps2	amplitude and phase signal for phase-shifter PS **???**	**?**

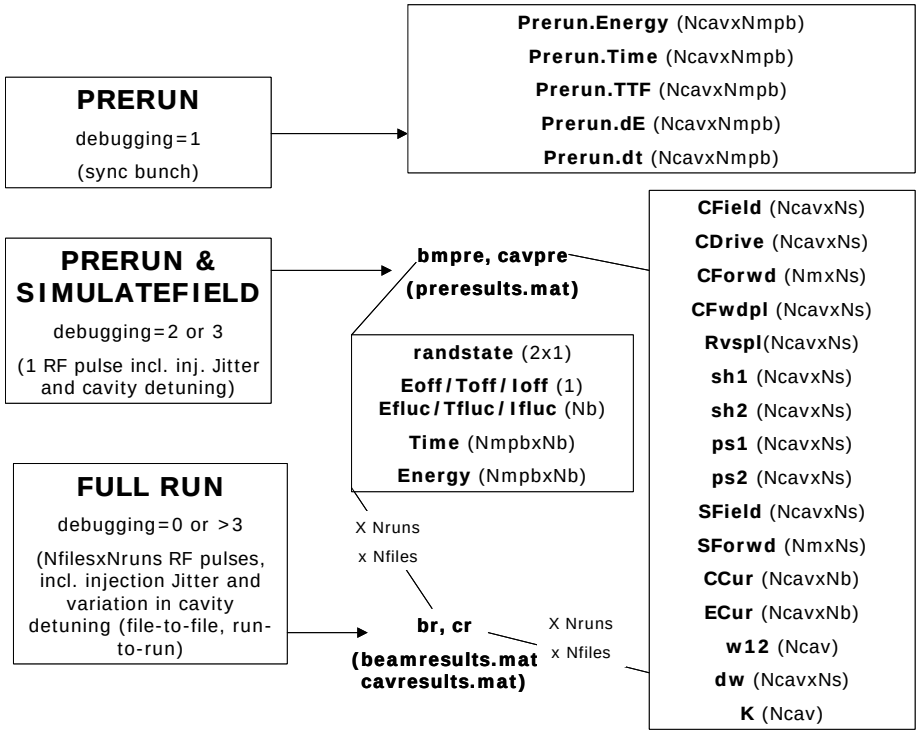


Figure 4-2: Output structures for the different program options.

and *cavityresults.mat* data files. MATLAB's load command makes the data available. They contain similar fields as those listed in Table 4-3. The only difference is that each run and file is a separate matrix with all *NcavxNs/Downsample* data (e.g. *cr(10).CField* is the cavity field matrix (*NcavxNs/Downsample*) for the 10th run). The reduction in the number of columns (or time steps) by the *Downsample* factor is described further in the context of **SimulateField.m.** Fig. 4-2 also shows the different output fields.

4.3 LoadInput.m

The **LoadInput.m** routine reads the input file linac.csv. It converts the linac.csv input into the MATLAB structures *Cavities*, *Phaseloop* and *Bunches*. In addition it produces a derived structure called *Mod*, which contains relevant information on the modules (Table 4-4). **Loadinput.m** also produces some additions to the *General* structure (Table 4-5). Finally this script also modifies some of the existing input data (such as converting phase from degrees to rad).

Table 4-4: Fields in the *Mod* structure-variable.

Field	Comment	unit
Cavities	string with indices of cavities contained in module (*Nm*)	-
N	number of cavities per module (*Nm*)	-
Feedback	average feedback gain of all cavities in module (*Nm*)	proportional
FillOff	average fill delay of all cavities in module (*Nm*)	μsec

Table 4-5: Additions to the *General* structure.

Field	Comment	unit
doPhaseloop	=1 if *Phaseloop* exists (*scalar*)	–

Fig. 4-3 shows a flow-diagram for **Loadinput.m**. The following discusses the main steps of the routine.

After checking that the data directory name is defined (and setting it to run0data if it is not defined) **LoadInput.m** opens linac.csv with the *fopen* function. The input file is read line by line with the *fgetl* function. When the '*{*' character, which signifies the beginning of a structure, is detected, a more complicated line-by-line data transfer is initiated. This line-by-line transfer consists in preparing the data (removing quotes, comments and unnecessary commas) for the *sscanf* function, which parses the strings from the *tline* variable to the temporary *A* matrix. Once the string '}' is detected, the data scanning is terminated and the data are parsed into a structure variable named *Varname,*

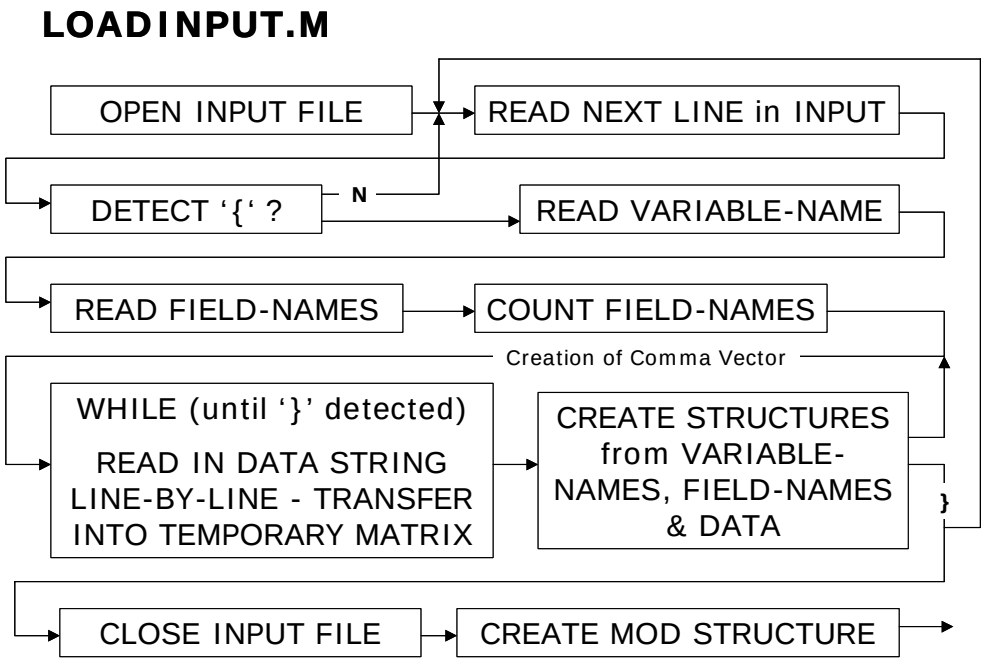


Figure 4-3: Flow-chart for LoadInput.m.

which contains the data in the respective sub-arrays *Fieldnames*. This process is repeated for as many times as there are "{" (three in our case).

LoadInput.m also prepares some additional structure variables from the input. These are the *Mod* and additions to the *General* structure. The *Mod* structure contains information about the number of cavities per module (and their indices), the average feedback gain and fill-off settings of all cavities in each module. The addition to the *General* structure consists in defining the scalar *doPhaseloop* variable, which is 1(0) if the *Phaseloop* structure is (not) defined in the input. The **LoadInput.m** file also defines the Lorentz-detuning constant *KL* (*-1 Hz/(MV)*^{2}), Lorentz-detuning ΔkL (0.1, relative to *K*) if they are not defined in *Cavities*. Note that the program assumes that the Lorentz-detuning factor, which is usually given in *Hz/(MV/m)*^{2}) was already divided by the square of the cavity length in the input. The *Cavities.Atten* field is created, with the default values 1. This field is not used anymore (historical artifact). **LoadInput.m** converts the phase advance setting in *Cavities.Phase* from degrees to rad. Finally **LoadInput.m** recalculates the *Cavities.Feedback* vector, dividing the gain factor given in the input by the number of cavities in each module and re-assigning it to each cavity accordingly.

4.4 Prerun.m

The **Prerun.m** routine simulates the passage of a pilot-particle and a pilot bunch consisting of *Nmpb* macro-particles through the linac. It essentially serves as a debugging tool. It also calculates the footprint of the synchronous particle in *E/t* phase-space. Fig. 4-4 gives a flow-chart for **Prerun.m**. Table 4-6 lists the output arrays.

Table 4-6: The output arrays of the Prerun.m routine are grouped into the Prerun structure. The fields within the structure are listed in the table.

Array	Comment
Prerun.Energy	energy of all mps in synchronous bunch along linac (*NcavxNmpb*)
Prerun.Time	arrival time of all mps in synchronous bunch along linac (*NcavxNmpb*)
Prerun.TTF	norm. trans-time-fact. of all mps in sync. bunch in all cavities (*NcavxNmpb*)
Prerun.dE	difference betw. mp energy and sync. energy in all cavities (*NcavxNmpb*)
Prerun.dt	difference betw. mp arrival time and sync. time in all cavities (*NcavxNmpb*)

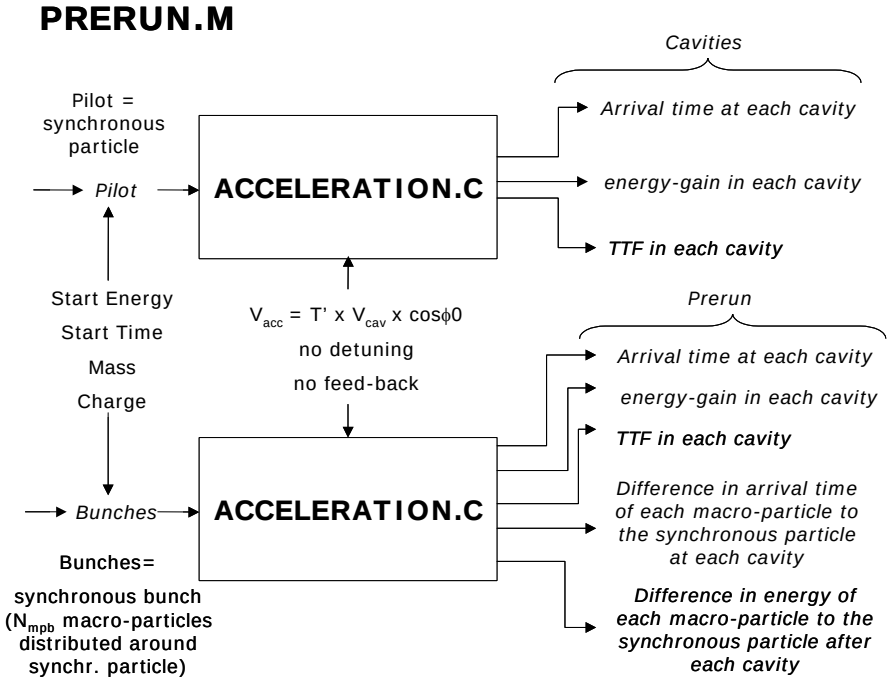


Figure 4-4: Flow-chart for Prerun.m.

Prerun.m first creates the structure variable *Pilot*, which contains several scalar fields, such as *Energy* (starting energy *Eini0*), *Time* (the arrival time of the pilot particle in the first cavity *tini0*, here automatically set to zero), *Mass* (938 MeV for protons), *Charge* (in units of the elementary charge), *N* (=1) and *I (*=1*)*. The current is irrelevant for the tracking of the synchronous particle since no beam-

loading is taken into account. The *Pilot* fields are for the most part initialized with the first line of the input structure variable *Bunches*.

Following the initialization of *Pilot*, **Prerun.m** calls **acceleration.c**, which simulates a run of the pilot particle through the linac. The variables of **acceleration.c**, however, are such that only one macro-particle with the above characteristics (*Pilot*), which is at the synchronous phase in all cavities, is assumed (instead of *Bunches*, *Pilot* is passed on in the function call). Also, the fields in the cavities are assumed to be the nominal design fields (*Cavities.Amplitude* is passed on in the function call). Since the pre-set beam phase-advance is taken into account in the calculation the pilot particle is, strictly speaking, not at the synchronous phase. For the purpose of all following calculations, however, this will be the reference (or "synchronous") phase since the beam phase advance is always applied. **Acceleration.c** returns the structure variable *ar*, which contains the *Time*, *Energy* and *TT*F arrays. All three arrays have *Ncav* (number of cavities) lines and *Nmpb* (number of macro-particles) columns. Since *Nmpb=1* for *Pilot*, they are vectors in this case. The time vector contains the arrival times of the pilot particle at the center of each cavity. The energy vector contains the energy of the pilot particle at the end of each cavity. The *TTF* vector contains the transit-time factor for the synchronous particle in each cavity. **Prerun.m** then uses the arrival time vector *ar.Time* to overwrite the *Cavities.Time* vector with the arrival times of the synchronous (or reference) particle as needed in subsequent sections of the program.

The second part of **PreRun.m** includes a simulation of an entire bunch with **acceleration.c**. The input to **acceleration.c** is such that only one bunch consisting of *Nmpb* macro-particles is accelerated. This bunch uses macro-particles distributed in *E/t* space such as prescribed in *Bunches*, but does not vary the bunch-center position from *Eini0/tini0*. It is therefore a synchronous bunch. The cavity amplitudes are again set to their nominal values, i.e. no beam-loading or detuning phenomena are taken into account (**SimulateField.m** is not called). As before the beam phase-advance is taken into account. The output of this second part of **PreRun.m** consists of the *prerun* arrays: *Energy*, *Time*, *TTF*, *dt* and *dE*. These are matrices containing the calculated quantities of the same name as a function of cavity (rows) and macro-particle (column). The first three are similar to those in *ar*, but they contain the $E_{j,k}$, $t_{j,k}$, $T'_{j,k}$ information for each macro-particle in the synchronous bunch. In the first column is the information about the first and thus synchronous macro-particle. The arrays *dt* and *de* are derived *NcavxNmpb* arrays, which contain information about the

arrival time difference and energy difference in each cavity of each macro-particle (in the synchronous bunch) to the synchronous particle.

4.5 acceleration.c

Acceleration.c tracks the macro-particles through the linac, i.e. it calculates sequentially the arrival time of the particle in each cavity. To obtain the arrival time in each cavity, **acceleration.c** needs to calculate the effective acceleration of each particle in every cavity. This calculation uses the transit-time factor (*T'*), the phase difference of the particle with respect to the synchronous particle ($\Delta\phi$) and the total, complex accelerating voltage ($\hat{V}$) in each cavity as provided in the function call. In the program **SimulateField.m** the cavity field calculation, taking into account beam-loading, detuning and vector-sum as well as vector-modulator feedback, is performed and passed on to **acceleration.c** for tracking. **Acceleration.c** also uses the

Table 4-7: The output arrays of the acceleration.c routine are grouped into the ar structure. The fields within the structure are listed in the table.

Array	Comment
ar.Energy	energy of all macro-particles along linac (*NcavxNmpb*)
ar.Time	arrival time of all macro-particles along linac (*NcavxNmpb*)
ar.TTF	normalized transit-time-factor of all mps in all cavities (*NcavxNmpb*)

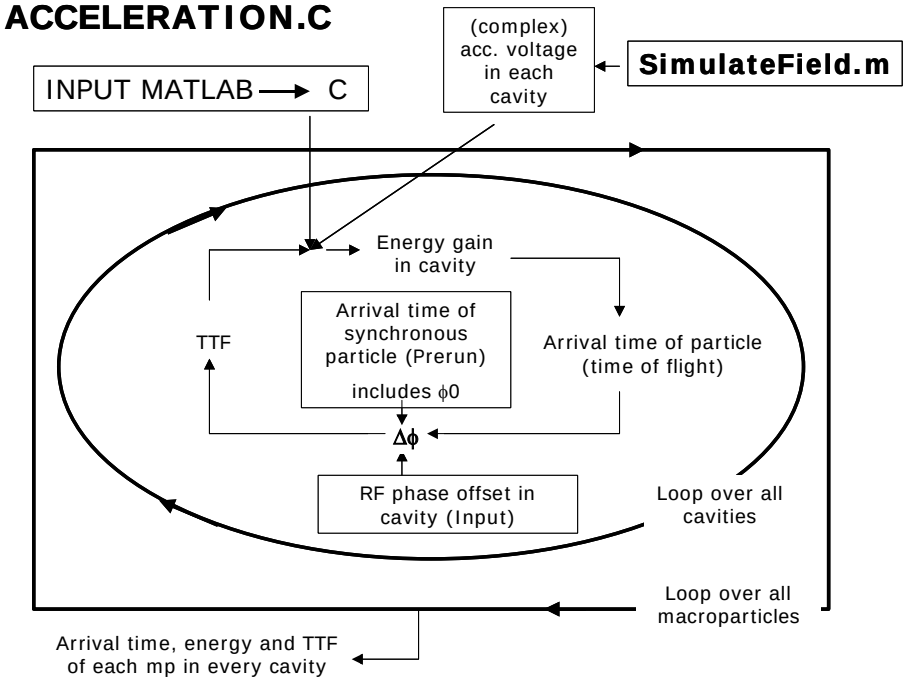


Figure 4-5: Flowchart of acceleration.c.

Cavities and *Bunches* structure as input parameters. Finally it also uses the *Energy, Time* and *TTF* vectors, calculated by **Prerun.m**, for the respective parameters of the synchronous particle. The output of **acceleration.c** consists of the *NcavxNmpb* vectors *Time*, *Energy* and *TTF*. They are summarized in Table 4-7. Fig. 4-5 shows the flowchart schematic for this subroutine. For faster computation **acceleration.c** is written in C-code. The **acceleration.c** routine can be compiled with the MATLAB mex command (producing *acceleration.dll*). In the Windows based program version *#include <math.h>* needed to be added in the file header.

The first part of the program defines C-arrays, which read in the required input parameters from the MATLAB variables produced by **LoadInput.m**. Most of these parameters are from the *Cavities* input field: -1- the position of the center of each cavity along the linac (*Position*), -2- the so-called *Time* vector (see discussion in section 4-4), -3- the cavity frequency *(Frequency*), -4- the phase-advance in each cavity (*Phase*), -5- the cavity βc (*Beta*), -6- the ratio of cell length and RF wave-length (*GapLambda*), -7- the number of cells per cavity (*Cells*), -8- the multi-cell mode in which each cavity is operated (e.g. π-m*ode*), -9- the energy (as calculated in the input for the synchronous particle), and, -10- the particle mass, -11- the particle charge and –12- the number of bunches.

Acceleration.c calculates the arrival time, $ta_{j,k}$, of each macro-particle *k* in each cavity *j* from the arrival time in the preceding cavity and the time of flight from the preceding cavity at the velocity given by the particle βb in the preceding cavity. The distances between two neighboring cavities are always measured from center-to-center. The center positions of the cavities are as provided in the input. The initial parameters $Eini_k$ and $tini_k$ vary with each macro-particle *k* and are provided by the input ($E0_k$, $t0_k$) and **SimulateField.m** (coherent and incoherent fluctuations). Eq (4-2) calculates the arrival time for a single macro-particle *k* in each cavity *j*.

$$ta_{j,k} = \sum_{n=0}^{j-1} ta_{n,k} + \frac{z_j - \sum_{n=0}^{j-1} z_n}{\beta b_{j-1,k}\, c} \qquad (\mathrm{sec}) \qquad (4\text{-}2)$$

The particle velocity βb is calculated from the particle energy with:

$$\beta b = \sqrt{1 - \frac{1}{\gamma^2}} \quad \gamma = \left(1 + \frac{E}{mc^2}\right),$$

where *E* is the kinetic energy of the macro-particle of given charge (*q*) and mass (*m*) contained in each macro-particle. Note that for the purpose of the calculation of the beam acceleration the exact number of single particles contained in a macro-particle is irrelevant. **Acceleration.c** therefore treats macro-particles as single particles (*q=Bunches.Charge*=charge of single particle, *m=Bunches.Mass*=mass of single particle in MeV). The kinetic energy of the macro-particle $E_{j,k}$ is calculated iteratively through the linac from the energy gain in each cavity *j*. The energy gain per cavity is calculated from the transit-time factor $T'_{j,k}$, the total cavity voltage $\hat{V}_j$ (in MV) and the phase difference to the synchronous particle $\Delta\phi_{k,j}$ according to Eq. (4-3). Note that there is a distinct difference between the phase difference between the beam and the synchronous phase ($\Delta\phi_{k,j}$) and the phase factor in the cavity field ($\phi c = \arctan \Im V / \Re V$), which is the result of a mismatch between RF power supply and the (detuned) cavity resonance frequencies. The later is taken into account in the amplitude and phase of the accelerating voltage. The effective (real) acceleration, given the phase of the cavity field $e^{i\phi c}$ and the beam phase $e^{i\phi b}$, is the real part of $Ve^{i(\phi c+\phi b)}$, which is $Vcos(\phi c)cos(\phi b)-Vsin(\phi c)sin(\phi b)$. $Vcos(\phi c)$ and $Vsin(\phi c)$ are obviously the real and imaginary parts of the complex cavity accelerating voltage $\hat{V}$.

$$\begin{aligned} \boldsymbol{E}_{j,k} &= \boldsymbol{E}0_k + \sum_{u=1}^{j} \Delta \boldsymbol{E}_{u,k} \qquad (\boldsymbol{MeV}) \\ \Delta \boldsymbol{E}_{j,k} &= \boldsymbol{qT'}_{j,k}\left(\beta c_j, \beta b_{j-1,k}\right)\left[\mathbf{Re}\hat{V}_j \cos\left(\Delta\phi_{j,k}\right) - \mathbf{Im}\hat{V}_j \sin\left(\Delta\phi_{j,k}\right)\right] \end{aligned} \tag{4-3}$$

Note that $\hat{V}$ includes the effects of beam-loading and cavity detuning (and the respective corrective actions by vector-sum control and fast vector-modulator feedback).

The field amplitudes in the cavities are calculated by the **SimulateField.m** sub-routine, which includes the simulation of beam-loading, detuning and feedback. The phase-difference between the particle *k* in cavity *j*, $\Delta\phi_{j,k}$ and the RF synchronous phase is calculated with:

$$\Delta\phi_{j,k} = \phi 0_j - \omega RF_j \Delta ta_{j,k} = \phi 0_j - \omega RF_j \left(tsync_j - ta_{j,k}\right) \qquad (rad), \tag{4-4}$$

where $tsync_j$ is the arrival time of the synchronous (or reference) particle in the center of cavity *j*, as calculated with **acceleration.c** during the first call by the **Prerun.m** routine (using the *Pilot* particle).

Note that *tsync* already includes the beam phase advance $\phi 0$, as read from the input (*Cavities.Phase*). The *tsync* vector is called *BeamTime* in **acceleration.c** (which is the new name of the *Cavities.Time* vector after the processing by **acceleration.c**). Before the simulation of the pilot particle, the *BeamTime* vector components are usually all (-1), as defined in the input file.

Note that the phase-mismatch ($\omega_{RF}(t_{sync}-t_a)$) of the particle in each cavity is not computed with respect to some absolute phase, but to the phase of the synchronous particle! This therefore assumes that the designer of the linac has already defined the RF phase-settings for all cavities that satisfy the basic time of flight delays between cavities of the synchronous particle. These constant phase-settings are obtained using module-to-module delays in the modulator pulse, variations in the length of wave-guide between klystron and individual cavities and three-stub mechanical tuner settings for the fine tuning of the forward power to individual cavities.

The transit time factor of the cavity, $T'_{j,k}$ used in Eq. (4-3), describes the effect of the beta-mismatch between beam and cavity on the effective acceleration voltage. In multi-cell cavities the transit time factor is dominated by cell-to-cell effects. The particular transit time factor implementation used here (Eq. (4-5)), referred to as *T'*, is slightly different from that commonly used in literature. It is normalized such that its maximum is 1 rather than 0.5. In this way the product $Re\hat{V}\,T'$ gives the effective accelerating voltage for the synchronous particle, independently of the particular longitudinal field profile of the cavity. (Strictly speaking, however, the transit time factor *T* given above only applies to the case of cavities with a more or less sinusoidal axial field profile, such as for instance in elliptical cavities, operated in the π mode). The transit time factor given below applies to a multi-cell cavity (*Nc* is the number of cells), where each cell is *Lc* long. The first term describes the single cell effects. The second term describes the cell-to-cell effects. The nominator in the first term was developed into a Taylor-series to prevent a singularity in the calculation. The second term also has a singularity at $\beta c=\beta b$, but this appears to be without consequences (probably because it is unlikely that $\beta c=\beta b$, both having a huge number of digits). A complete derivation of the transit-time-factor can be found in appendix A.

$$T' = 2T = 2\frac{\left[1-\left(\frac{\pi^2}{24}\right)\left(1-\frac{\beta_c}{\beta_b}\right)^2+\left(\frac{\pi^4}{1920}\right)\left(1-\frac{\beta_c}{\beta_b}\right)^4\right]}{\left(1+\frac{\beta_c}{\beta_b}\right)}\frac{\sin\left[\frac{N_c\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}{N_c\sin\left[\frac{\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]} \qquad (4\text{-}5)$$

The passage of the macro-particles through the linac is simulated in two *for*-loops. The outer loop is over the macro-particles (*nbunch* as running index, which is an unfortunate choice as name) such as ordered in *Cavities.Bunches*. The inner loop is over the cavities (n*cav*). This inner loop calculates (and stores) the actual energy of the macro-particle and their arrival time sequentially, cavity-by-cavity.

4.6 SimulateField.m

SimulateField.m calculates the cavity fields as a function of time, including beam loading, detuning and feedback for all bunches (and all its macro-particles) contained in one RF (macro-) pulse. Feedback loops using vector-sum regulation at the RF module level as well as individual cavity vector-modulators are simulated. This simulation obviously also includes the feed-forward settings used to pre-compensate for detuning and beam loading. The calculation of the cavity detuning as a result of Lorentz-forces and microphonics (fast and slow) including the feed-forward, is in part provided by the separate **detuning.m** routine, which is discussed later. The effect of the vector-modulator is also modeled separately in the **dophaseloop.m** script, which is called by **SimulateField.m**. **SimulateField.m** uses the **acceleration.c** routine for the calculation of the beam acceleration, since the phase and energy of the macro-particles also affect beam-loading. Finally it also uses the special **dimsum.c** routine as part of the simulation of the vector-sum feedback.

SimulateField.m receives the *Cavities*, *Bunches* and *General* input arrays as read from the input file with the **Loadinput.m** routine. It also uses the *Time* and *Energy* vectors for the synchronous particles, as provided by **Prerun.m**. In addition it uses the *Mod* structure, provided by **Loadinput.m**, which contains information on each RF module as it is relevant for the RF phase and amplitude regulation at the klystron/module level (vector-sum control). The information contained in the *Phaseloop* array is needed for the control of the

feedback at the cavity level using vector-modulators. At this point, however, most of the *Phaseloop* fields are not used (with the only exception of the regulator gains) and most parameters of the feedback loops are hard-coded into the program. The use of *Phaseloop* will allow for individual adjustment of the phase-shifter feedback circuits (gains, initial set values, saturations, delays, time constants,..etc).

The output of **SimulateField.m** consists of the *cr* (cavity-results) and *br* (beam-results) files, which contain the information on the fields in the cavities and the longitudinal bunch dynamics. The program makes a difference between the first and the subsequent runs. The first run is stored in the *preresults.mat* datafile (created automatically by the program in *datadirectory*). The subsequent runs are stored in the *cavresults.mat* and *beamresults.mat* files. The beam-result structure, *br*, consists of the (final) *Energy*, (arrival) *Time* and *I* arrays, which have as many rows as there are macro-particles and as many columns as bunches/time-steps (with beam on). One of each of these structures is stored for each run (pulse). Also stored are the coherent (scalar) and incoherent (vector, with an element for each bunch) *Eini,tini,Iini* contributions. These structures are initialized with the respective values in the *Bunches* field from the input file.

The main field in the cavity-result structure, *cr*, is *CField*, which contains the cavity accelerating voltages for every bunch. The amplitude and phase changes as a result of detuning and beam-loading are also included. Since the number is complex the phase information is contained in the phase of the complex number:

*Amplitude(MV)=abs(CField), Phase(deg)=180/pi*angle(CField)*

The *CForwd* field contains the feedback voltage correction signal in $(MW)^{0.5}$ (incl. feed-forward) after vectorsum control. More precisely *CForwd* is the amplified difference between the set voltage and the actual voltage plus the feed-forward component *SForwd*. It is therefore the vector-sum control signal and proportional to the voltage that the feedback system tries to impose in the cavity. Squared it gives the beam-power that is demanded from the klystron to bring the field to the desired level. Note that the program, unlike the real case, does not pre-compensate for the reflection and attenuation of the forward power as a result of detuning and coupling loss. These effects (as discussed in appendix B) reduce the actual power the cavity receives. Since the program includes a strong feedback component, these losses will be made up by an increased demand in power during the next time step. Since the vector-sum feedback is performed at the module

level, the *CForwd* array has the dimension *NmxNs*, Note that $CForwd_{l,i}$ is instantaneous, for each time step *i* (i.e. it is not integrated!). This implies that the RF power source can follow instantaneously to load changes.

The *CDrive* field is the amplified difference between the set voltage and the actual voltage plus the feed-forward component *SForwd* as well as the phase-shift and attenuation provided by the phase-shifter, already converted to MV. In the current implementation it also takes into account multiple power reflection between the cavity and the phase-shifter. *CDrive* therefore is the feedback control signal after the vector-modulator in MV. The *CDrive* array has dimension *NcavxNs* (although not all cavities might have vector-modulators, in which case the phase-shifter effect is 1). Note that all these fields are complex, with the real part containing the amplitude information and the imaginary part holding the phase information. The cavity reference phase is zero, therefore all phase errors are with respect to zero and the feedback/feed-forward systems therefore aim at restoring zero phase. In a real machine, of course, the cavity reference phase would include the synchronous phase, cable delays,..etc.

CFwdpl holds the actual forward signal supplied by the fast ferrite vector-modulators (complex, i.e. amplitude, relative to 1, and phase in rad). Since the set-point phase of the phase-shifters is usually not zero, amplitude signals >1 are possible. *Rvspl* contains the reflection signal from the vector-modulators (relative to 1).

Sh1 and s*h2* contain the phase correction signals (rad) provided by the two branches of the phase-shifters.

Ps1 and *ps2* provide the phase-shifter power supply signals (they are presently not used!) as determined by the program.

The remaining output arrays are *w12* (cavity bandwidth) and *dw* (total cavity detuning phase) as well as *K* (Lorentzforce detuning constant for each cavity). These can be useful to calculate parameters related to the detuning. The detuning is obviously also an important output parameter. These arrays are listed in Table 4-8 and 4-9. Fig. 4-6 shows a flow-schematic for **SimulateField.m**. After the first run (debugging=2 or 3) the resulting structures are called *bmpre* and *cavpre*.

Table 4-8: The arrays of the beam-results structure (br).

Array	Comment	units
randstate	state of MATLAB randn function (*2 scal*)	-
Eoff	initial sigma E offset of bunch centroid – coherent part (*scal*)	MeV
Toff	starting time offset of bunch centroid - coherent part (*scal*)	sec
Ioff	variation bunch current - coherent part (*scal*)	A
Efluc	initial sigma E offset of bunch centroid – incoherent part (*Nb*)	MeV
Tfluc	starting time offset of bunch centroid - incoherent part (*Nb*)	sec
Ifluc	variation bunch current - coherent part (*Nb*)	A
Energy	final energy of macro-particles for all bunches (*NmpbxNb*)	MeV
Time	arrival time at last cavity of mp for all bunches (*NmpbxNb*)	sec

Table 4-9: The arrays of the cavity-results structure (cr).

Array	Comment	units
CField	actual, complex accelerating voltage in each cavity at each time step (*NcavxNs*)	MV
CForwd	instantaneous, amplified difference signal between set-voltage and actual voltage (vector-sum control signal) (*NmxNs*), gives klystron power when squared	$MV/\Omega^{0.5}$
CDrive	incremental, forward voltage supplied for each cavity at each time step (*NmxNs*)	MV
CFwdpl	actual, complex forward signal (complex) from vector-modulators at each time step (*NcavxNs*)	rel. (rad)
Rvspl	actual, complex reflection from vector-modulator (*NcavxNs*)	rel
SField	set-point (real) accelerating voltage in each cavity at each time step (*NcavxNs*)	MV
SForwd	incremental, complex feed-forward set-point forward power to each cavity at each time step (*NmxNs*)	$MV/(\Omega^{0.5})$
sh1	actual phase-shifter amplitude correction (*NcavxNs*)	rel.
sh2	actual phase-shifter phase correction (*NcavxNs*)	rad
ps1	actual phase-shifter PS signal for amplitude correction (*NcavxNs*)	?
ps2	actual phase-shifter PS signal for phase correction (*NcavxNs*)	?
CCur	actual, complex beam-loading factor (*NcavxNs*)	A
ECur	actual, average difference of bunch energy to synchronous particle (*NcavxNs*)	MeV
w12	cavity bandwidth, HWHM (=π*fRF/QL*) (*Ncav*)	rad-Hz
dw	actual, total cavity frequ. shift due to Lorentzforce and microphonics detuning and feed-forward (*NcavxNs*)	rad-Hz
K	Lorentz-force detuning constant for each cavity, fixed for entire pulse (*Ncav*)	$Hz/(MV)^2$

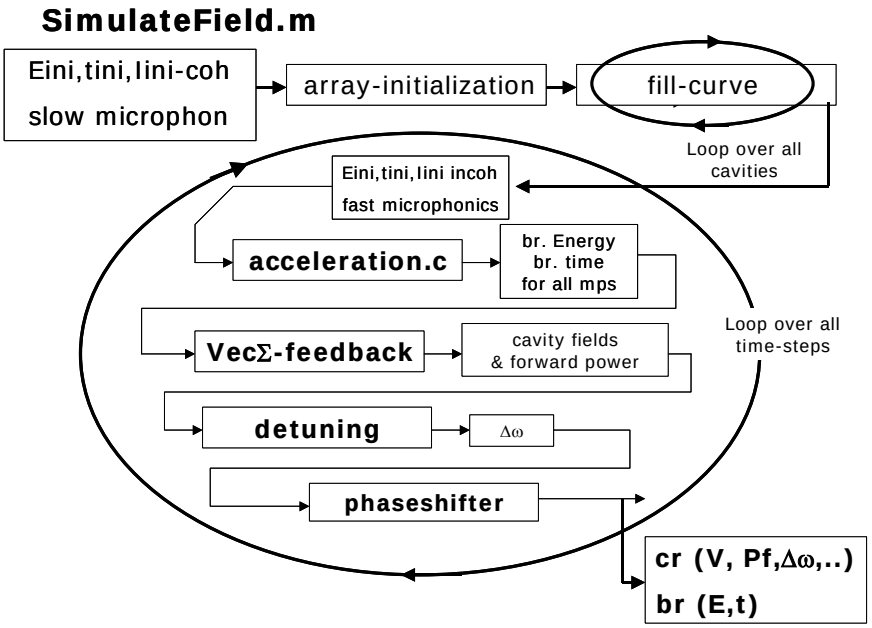


Figure 4-6: Flow diagram for SimulateField routine.

Injection Jitter

The starting-energy, starting-time and current ("0") of the macro-particles for the synchronous bunch (i.e. the bunch with a synchronous centroid) are defined in the *Bunches* field in the input (see discussion in **3.5**). In addition, the bunch centroid is shifted randomly to simulate beam jitter at injection. There are coherent ("*coh*", pulse-to-pulse) and incoherent ("*fluc*", bunch-to-bunch) contributions to the injection energy, time and bunch-current jitter. The coherent contribution is added at the start of **SimulateField.m**. The incoherent contribution is added within the loop over the bunches (=time-steps with beam on) within a pulse. The coherent and incoherent contributions are calculated through multiplication of the MATLAB *randn* function[1], which produces a (pseudo) random number from a normal distribution around zero (variance 1), with the respective fluctuation sigma from the input field *General*. The centroid position in longitudinal phase-space at the end of one pulse (after iteration through as many bunches as there are "beam-on" time steps within a pulse) becomes the initial value for the next pulse. Depending on the number of

[1] The two implementations used in SimulateField.m are randn and randn(Nbeam,1). The former produces a random number and is used for the coherent contribution. The latter produces a vector with Nbeam random numbers and is used for the incoherent fluctuation. Nbeam is the number of time-steps during beam-on time and thus the number of bunches in a pulse. The sequence of numbers generated is determined by the state of the generator. Since MATLAB resets the state at start-up, the sequence of numbers generated will be the same unless the state is changed.

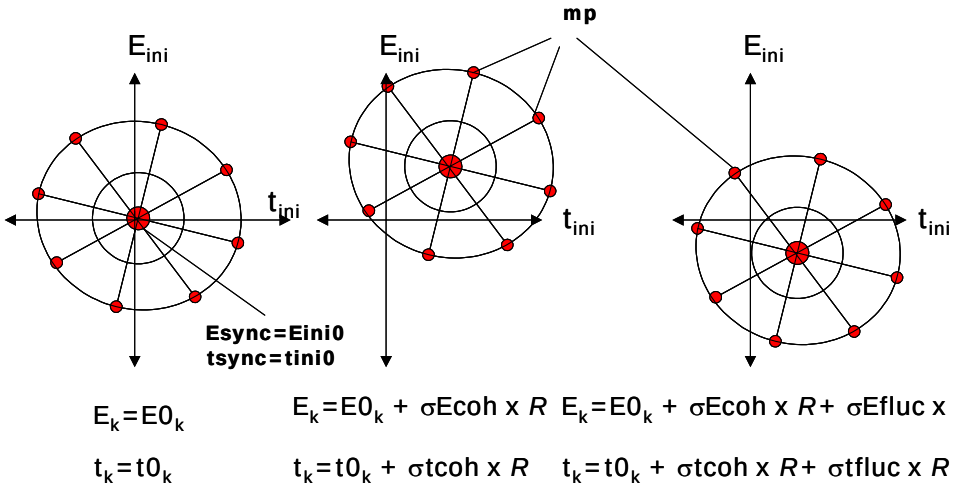


Figure 4-7: The different levels of variations of the macroparticle (k) position in longitudinal phase-space.

arguments the *randn* function will return different results. The initial state of *randn* at the start of **SimulateField.m** is stored in the *br.randstate* variable from where it can be retrieved to recalculate a particular case. In that case, the *randstate* variable needs to be added as a fifth input variable to the call of **SimulateField.m**. Therefore **SimulateField.m** first checks for this fifth variable and uses it to set the state in the *randn* function. The *randn('state')* is a two element vector. Fig. 4-7 illustrates the different steps for the definition of the beam longitudinal phase-space jitter at injection.

Eq. (4-6) gives the complete formula for the start value settings of the mps in longitudinal phase space. The bunch-to-bunch variation of the incoherent contribution is symbolized in Eq. (4-6) by the index *i*.

$$\boldsymbol{Eini}_{k,i} = \boldsymbol{E}0_k + \boldsymbol{\sigma Ecoh} \times \boldsymbol{R} + \boldsymbol{\sigma Efluc} \times \boldsymbol{R}_i \quad \textbf{... } \boldsymbol{i} = 1,..,\boldsymbol{Nb} \qquad (\boldsymbol{MeV}) \qquad \text{(4-6a)}$$

$$\boldsymbol{tini}_{k,i} = \boldsymbol{t}0_k + \boldsymbol{\sigma tcoh} \times \boldsymbol{R} + \boldsymbol{\sigma tfluc} \times \boldsymbol{R}_i \quad \textbf{... } \boldsymbol{i} = 1,..,\boldsymbol{Nb} \qquad (\textbf{sec}) \qquad \text{(4-6b)}$$

$$\boldsymbol{Iini}_{k,i} = \boldsymbol{I}0_k \left(1 + \frac{\boldsymbol{\sigma Icoh}(\%)}{100} \times \boldsymbol{R}\right) \left|1 + \frac{\boldsymbol{\sigma Ifluc}(\%)}{100} \times \boldsymbol{R}_i\right| \quad \textbf{... } \boldsymbol{i} = 1,..,\boldsymbol{Nb} \qquad (\boldsymbol{A}) \qquad \text{(4-6c)}$$

E0$_k$, *t0*$_k$ and *I0*$_k$ values are those defined in the input (*Bunches*). They obviously also include the *Eini0* and *tini0* offset of the synchronous particle.

Array Initialization

Simulatefield.m then proceeds to initialize the main calculation and output arrays in the *beamresults*, *br*(*NmpbxNb*), and *cavresults*, *cr*(*Ncav or Nm x Ns*), structures. These arrays are usually matrices with a row for each macro-particle / cavity (or module) and a column for each bunch / time-step. While in the case of the cavity parameters the data for each time step are stored, the number of time steps is limited to those during which there is beam in the case of the beam parameters. These arrays are listed in Table 4-8 and Table 4-9.

After array initialization, **SimulateField.m** prepares some parameters, which can be computed outside the time-loop. Among them is the cavity bandwidth $\omega 12\ (=\pi fRF/QL)$. Others are the cavity filling, the cavity detuning due to slow microphonics and the set-value for the RF power supplied to the cavity during filling and beam.

Filling Curve

The function describing the filling of each cavity uses a universal fill-curve, which depends only on the fill-time tf_j (and fill-time delay tfo_j) of each cavity j as defined in the input. As discussed in chapter 3 the individual fill-time of each cavity is determined from tf_j-tfo_j. The universal fill-function, $v(t)$, is relative to a cavity amplitude of 1 (a.u.). It is then multiplied with the nominal voltage $Vcav_j$ in each cavity to obtain the set-values of the cavity voltages $SField_j$. Eq. (4-7) gives $v_j(t)$, as it is calculated separately for every cavity j.

$$\boldsymbol{v}_{j,i} = \begin{cases} 0 & \boldsymbol{t}_i < \boldsymbol{tfo}_j \\ 2\left(1 - \boldsymbol{e}^{-\left(\frac{t_i - tfo_j}{tf_j - tfo_j}\right)\ln 2}\right) & \boldsymbol{tfo}_j < \boldsymbol{t}_i < \boldsymbol{tf}_j \\ 1 & \boldsymbol{t}_i > \boldsymbol{tf}_j \end{cases} \qquad \text{(4-7-old)}$$

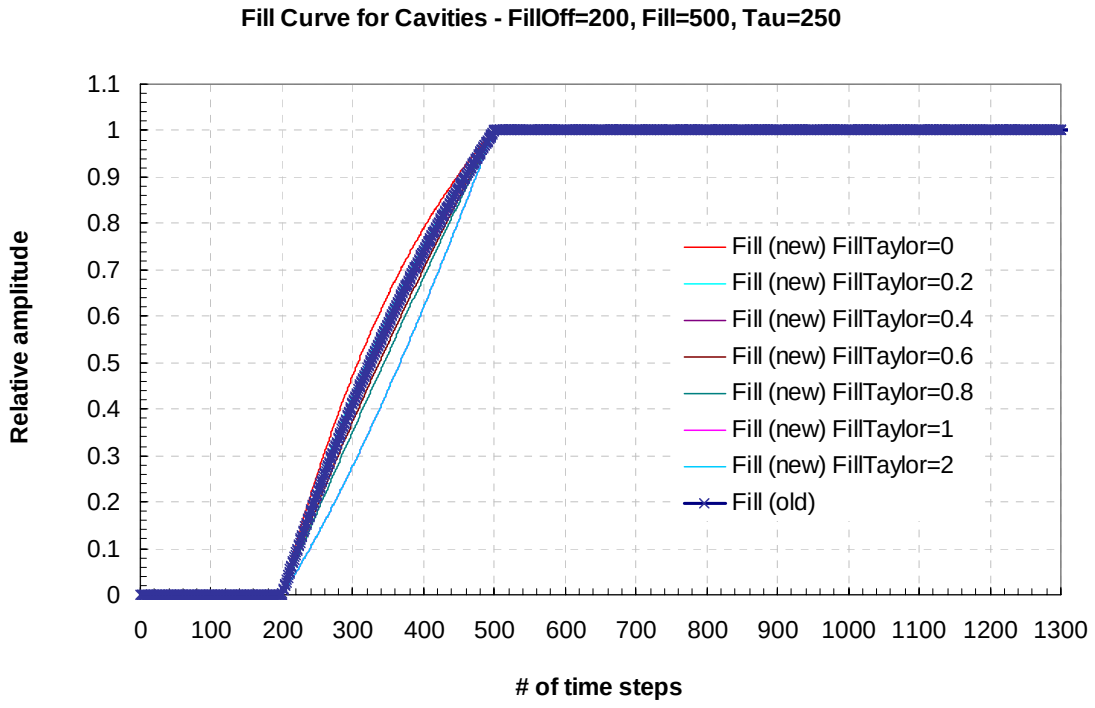


Figure 4-8: Cavity fill-curve for a t_{fill}=500 and a $t_{fill\text{-}off}$=200.

Also shown in Fig. 4-8 is a different filling function, which is used in the latest implementation of the program ("New filling function"). This new function includes the FillTaylor parameter, which allows varying the fill-function shape from linear (*FTayl=1*) to exponential (*FTayl≠1*). Since the *FTayl* factor is given in the input for every cavity, cavity-to-cavity variations of the filling function can be implemented. $N_{filloff}$ is the number of time steps during the fill-off delay.

$$v_{j,i} = \begin{cases} 0 & t_i < tfo_j \\ \frac{1}{nf_j}\left[\left(1-e^{-n_{i-Nfilloff,j}}\right)+FTayl_j\left[n_{i-Nfilloff,j}-\left(1-e^{-n_{i-Nfilloff,j}}\right)\right]\right] & tfo_j < t_i < tf \\ 1 & t_i > tf \end{cases}$$

$$n_{i,j} = 0, \frac{1}{\tau f_j}, \frac{2}{\tau f_j}, ..., \frac{i}{\tau f_j}, ...nf_j \quad \in (0, \sim 1), \quad nf_j = \frac{tf - tfo_j}{\tau f_j}$$

(4-7-new)

Forward Power Feed-Forward

The forward power to the module *l* allows setting or maintaining the field $Vcav_j$ during the filling and beam loading. In the notation used here, the feed-forward signal, *Sfwd*, has the unit square root of power. Eq. (4-8) gives the forward voltage feed-forward defined in **SimulateField.m**. The program applies the same philosophy as was applied to TTF, which is that of adaptive feedback, i.e. the feed-forward is chosen such as to minimize the feedback needed. Note that the index *l* indicates the average of all cavities *j* in the module *l*. This averaging is needed since the forward power supplied by the klystron is the same for all cavities in a module. Also note that the forward signal is obviously proportional to the forward power delivered by the klystron. It is also proportional to the accelerating voltage the cavity will deliver as it receives the forward power from the klystron.

As typical for the strongly over-coupled cases, which seek to obtain the matched cavity impedance with beam, the cavity is not matched during filling. In the strongly over-coupled condition the coupling losses strongly dominate the wall losses in the cavity: *RL~Rext~rQL*. The drive signal is typically two times the nominal accelerating voltage because of the strong reflection in the unmatched case (see appendix B for further explanation). Here it was reduced to 50% of the nominal voltage because the vector-sum feedback was found to be strong enough to regulate the voltage during filling. During beam-loading the cavity is matched and the forward voltage goes through to the cavity unimpeded. In this case the default forward power drive signal is chosen such as to compensate roughly for the expected voltage drop in the cavity due to beam-loading. In TTF the drive signal at flat-top is typically ~25% of the nominal voltage. Here the expected beam-loading is calculated directly, including the beam phase advance as the expected difference between the beam and the klystron phase. The feed-forward phase factor therefore neglects the phase variations caused by variations in the acceleration history of the bunch. This approach is good enough to first order, given that the variations from the phase advance offset are usually kept to the <5° level (with the help of the vector-sum regulation, and in some cases, the phase-shifters). The formula describing the beam-loading is discussed in further detail later.

Eq. (4-8) gives the instantaneous feed-forward signal to the cavities. It suffices to note that in Eq. (4-8), the 10^{-6} factor serves to scale the voltage to MV, while the $(1-\exp(-\omega_{12}\Delta t))$ factor can be understood when inserting *Sfwd0* into Eqs. (4-13) & (4-14). It is essentially a

factor, which will cancel out such as to compensate exactly the voltage drop due to beam-loading (such that $\hat{V}_{j,i+1} = \hat{V}_{j,i}$) THAT IS NOT TRUE (CHECK!!). In fact it anticipates the coupling loss in the cavity. Coupling loss is discussed in further detail in appendix B. It is important to note that the forward drive signal is instantaneous. *Sfwd0* is the initial matrix of the forward signal field referred to as *SForwd* in the program.

$$Sfwd0_{l,i} = \sqrt{Pfwd0_{l,i}} = \begin{cases} 0 & t_i < tfo_l \\ \dfrac{Vcav_l}{2\sqrt{2QL_l r_l}} & t_i < tf \quad \left(\dfrac{MV}{\sqrt{\Omega}}\right) \\ \dfrac{r_l\, Ib_i\, \omega RF_l\, e^{-i\phi 0_l}\, \Delta t\, 10^{-6}}{\left(1 - e^{-\omega 12_l \Delta t}\right)\sqrt{2QL_l r_l}} & t_i > tf \end{cases} \tag{4-8}$$

$$SForwd_{j,i}(MV/(\Omega)^{1/2}) = Sfwd0_{j,i}$$

The program includes the so-called conversion factor $\sqrt{2RL} \approx \sqrt{2rQL}$ to convert the drive signal into the square root of the forward power. Note that the drive signal should not only be squared, but also multiplied with 10^{12} to obtain the result in Watt, since the voltages are in MV. As will be explained later the forward signal is converted into a forward voltage increment (*CDrive*) before adding it to the actual cavity field (*CField*). *CDrive*, however, also includes the contributions of vector-sum and vector-modulator feedbacks, attenuations, feedback gains, ...etc.

Cavity Detuning

As discussed in further detail in appendix B, cavity detuning strongly increases the amount of power reflected from the cavity. Superconducting cavities, with their narrow bandwidth ($\omega 12 < 1$kHz,HWHM), are strongly affected by detuning. The cavity bandwidth, $\omega 12$, can be calculated with Eq. (4-9).

$$\omega 12_J = \pi \frac{fRF_j}{QL_j} \quad (rad - Hz) \tag{4-9}$$

When the cavity is detuned from the RF drive signal by $\Delta\omega$, the ensuing phase-shift is arctan($\Delta\omega/\omega 12$). When $\Delta\omega = \omega 12$ the phase-shift is 45°. The voltage amplitude is also reduced by reflection as a result of the frequency mis-match (at $\Delta\omega = \omega 12$ the voltage drops by a factor2).

Although detuning is a dynamic event resulting from mechanical excitation of the cavity, feed-forward correction can be applied to some extent. The Lorentz-force detuning, for instance, can be approximately anticipated, pre-detuning the cavity to some positive $\Delta\omega$ that will then compensate the Lorentz-force detuning at nominal field after filling. In TTF a Piezo tuner was successfully used to compensate the detuning using a feed-forward function. **SimulateField.m** applies the fixed pre-detuning approach. Such pre-detuning can be obtained by mechanically deforming the cavity. The pre-detuning approach results in a strongly detuned cavity at the beginning of the filling process, with the detuning gradually decreasing as the cavity deformation occurs during the filling.

SimulateField.m calculates the cavity detuning - see Eq. (4-10) – as a result of (listed in the same order as in the equation) Lorentz-force feed-forward pre-detuning (positive to compensate for the expected Lorentz-force detuning, which is negative), slow microphonics, fast microphonics and Lorentzforce-detuning ($\Delta\omega LF$). The Lorentzforce pre-detuning, $\Delta\omega L_j$, and the slow micro-phonics detuning $\sigma m_j R_j$ are both set during the initialization of $\Delta\omega$. Note that the **scream.m** script recalculates the pre-detuning from the new Lorentz-force detuning constants *KL* (see the discussion in 4.2). The fast microphonics detuning contribution, $\sigma mf_j R_j$ is added again at each time step (with $R_{j,i}$ varying each time *i*). The Lorentzforce detuning is field dependent and is therefore recalculated from the actual cavity fields for the next time-step. It is also the strongest contribution to Eq. (4-10). The calculation of $\Delta\omega LF$ is discussed in section 4.7.

$$\Delta\omega_{j,i} = \Delta\omega L_j + 2\pi\sigma m_{j,i} R_{j,i} + 2\pi\sigma mf_{j,i} R_{j,i} + \Delta\omega LF_{j,i} \quad (rad - Hz) \qquad (4\text{-}10)$$

$\Delta\omega L_j$ is usually positive and ~2 kHz in TESLA type cavities, several times larger than the cavity bandwidth. It is calculated in the input from the Lorentz-detuning constant and the nominal cavity voltage. **Scream.m** recalculates it to adjust it to the randomly varied individual cavity detuning constants.

The half-widths of the slow and fast distribution of microphonics effect in cavity *j*, σm_j and σmf_j is taken from the *Cavities* input. Since the slow micro-phonics detuning is updated every pulse its

characteristic time constant is ~Hz. The fast micro-phonics distribution is updated at every bunch (=time-step=1μsec), so the characteristic time constant is MHz. It is not clear whether such fast microphonics really exist. The fast micro-phonics was implemented nevertheless. The micro-phonics detuning frequencies being much smaller (10-100 times) than the Lorentz-force detuning frequency, it is secondary. When the Lorentz-force detuning is compensated, however, microphonics can become the leading detuning term (it remains much smaller than the bandwidth).

The Lorentzforce-detuning frequency shift is one of the last elements calculated in **SimulateField.m**. It is, in fact, calculated by a special routine called **detuning.m**, called from **SimulateField.m** once every time step for the next time step. The Lorentz-detuning needs to be recalculated at every time-step, because it depends on the cavity fields, which can vary from bunch-to-bunch. The Lorentzforce detuning constant K_j (Hz/(MV)^2) of the jth cavity is taken from the *Cavities* input.

The half-width of the distribution of Lorentz-detuning constants in cavity j, ΔkL_j , which is called *Kspread* in *Cavities* and defined relative to *KL*, is added to the Lorentz-force detuning constants once every "file" (that is why there is no bunch index *i* with the random function *R* in Eq. (4-9). The random variation of the Lorentz detuning constant for each cavity (the last term in Eq. (4-9)) is performed in **scream.m**, which prepares the new KL_j vector for the *Simcav* structure. This set of constants is kept for every *Nfile*. As before, the random numbers are provided by the MATLAB *randn* function. The $\Delta\omega$ factor for each cavity at each time step is also given in the output (*cr.dw*).

Loop over Time

At the core of the **SimulateField.m** routine is the calculation of the instantaneous field in the cavities during the passage of every bunch. Following initialization of the main parameter arrays and the beam injection jitter settings described above, **SimulateField.m** enters the loop over the time steps. **SimulateField.m** tracks the particles through the linac before calculating the cavity fields. The cavity field changes due to beam loading and feedback are therefore applied to the next time-step. The beam tracking is performed with **acceleration.c**, which calculates the energy-time profile of all macro-particles in the bunch along the linac. The bunch energy and especially the phase information are necessary for the subsequent calculation of the field amplitude in each cavity, *CField,* because of its effect on the

beam-loading. Once the bunch has been tracked through the linac the *Time* and *Energy* arrays (*Ncav*×*Nmpb*) are available to calculate the beam-loading parameter for the next bunch. The main parameters for beam-loading are the phase of the bunch with respect to the synchronous phase (which includes the beam phase advance!), the bunch current and the beam energy dependent (also in *Vfwd*) transit time factor. The phase-factor is calculated from the difference of the cavity arrival time and the *Cavities.Time* vector, which contains the arrival times at the cavities for the synchronized particle.

Cavity Voltage

Fig. 4-9 shows the basic physics model according to which the cavity voltage is calculated. This model assumes that the beam-loading (*⊿Vb)* occurs in a much shorter time than the duration of the time step (here 1 μsec), thereby approximating the beam-passage with a delta-pulse. After the passage of the bunch the refilling of the cavity, which is a slower process, brings the cavity voltage back to more or less the desired level. The cavity re-filling within a time step is assumed to occur at constant klystron power. At the same time the cavity is losing power through the input coupler. As clearly shown in the plot the filling and coupling loss functions are linear for the duration of a few time steps. Also, the total cavity forward power is regulated (feed-forward

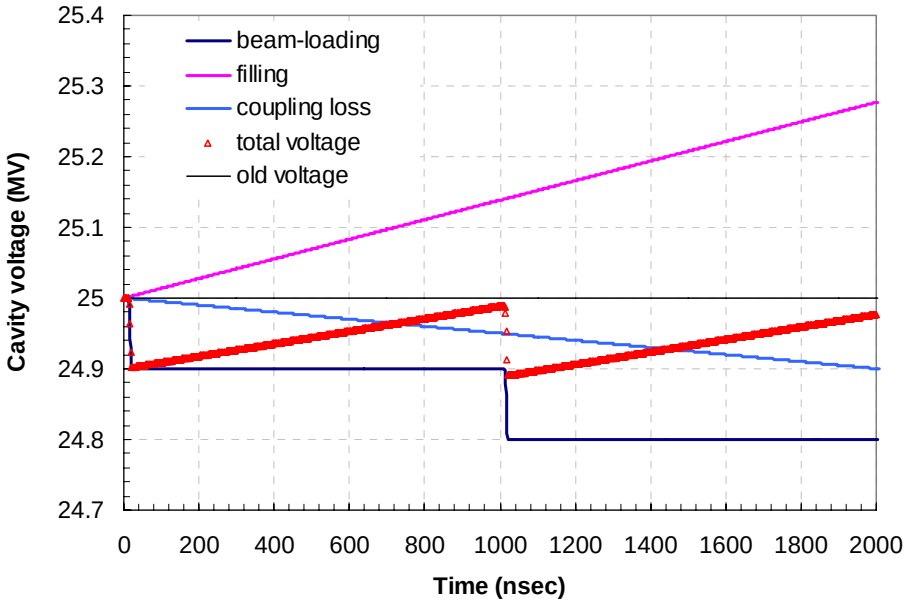


Figure 4-9: Example of beam-loading, filling, coupling loss and the resulting cavity voltage.

voltage *Vfwd)*. The regulation includes the tabulated feed-forward settings (*SFwd0*), the vector-modulator phase-change and amplitude (*CFwdpl*) and the vector-sum feedback. The vector-sum (included in *CForwd*) and vector-modulator (*CFwdpl*) feedback settings applied were determined during the previous time step. This process is repeated every time step. The change of voltage in the cavity is described with Eq. (4-11) [2], where $\Delta\omega$ is the cavity detuning (Eq. 4-10) from the klystron drive frequency and $\hat{V}_{j,i}$ the complex cavity field. The derivation of this equation is given in Appendix B.

In the Fermilab PD the bunch spacing is much shorter than 1 μsec, but the bunch charge in S.C.R.E.A.M can be chosen such that the average beam-loading is the same. The fact that the filling curve is linear within the time step of the program allows this simplification. (The voltage at the end would look the same if the beam-loading steps would be distributed over many smaller steps.)

SimulateField.m calculates the cavity fields and forward power settings for the next time-step (bunch) using Eq. (4-11), where $\hat{V}_{j,i}$ is the voltage in cavity *j* at time *i*, $\Delta Vb_{j,i}$, the beam loading voltage due to bunch *i*, and $Vfwd_{j,i}$, the forward wave voltage as supplied by the klystron via the phase-shifter (if there is one). This equation describes the subtraction of the beam-loading voltage (Eq. (4-12)) from the cavity voltage as well as the addition of the forward power. The factor $exp(-\omega 12\Delta t)$ describes the power lost through the input coupler. The phase factor $exp(i\Delta\omega\Delta t)$ describes the detuning of the cavity. The calculation of $\Delta\omega$ was discussed before (Eq. 4-10) and is discussed again in section 4.7, where the **detuning.m** script is introduced. The actual cavity voltage field $\hat{V}_{j,i} = CField_{j,i}$ is a complex matrix of dimension *NcavxNs*. The difference between the cavity phase and the klystron (or reference) phase (=zero) is contained in the complex angle. Referring to the graphs in Fig. 4-9 this calculation provides the voltage at every μsec (the end points of the shown graphs). Also given below is Eq. (4-11) in the notation used in the program.

$$\begin{aligned}\hat{V}_{j,i+1} &= \left\{\hat{V}_{j,i} - \Delta Vb_{j,i} - Vfwd_{j,i}\right\} e^{-\left[\left(\omega 12_j - i\Delta\omega_{j,i}\right)\Delta t\right]} + Vfwd_{j,i} = \\ &= \left[\hat{V}_{j,i} - \Delta Vb_{j,i}\right] e^{-\left[\left(\omega 12_j - i\Delta\omega_{j,i}\right)\Delta t\right]} + Vfwd_{j,i}\left[1 - e^{-\left[\left(\omega 12_j - i\Delta\omega_{j,i}\right)\Delta t\right]}\right](MV)\end{aligned} \qquad (4\text{-}11)$$

$$CField_{j,i+1}(MV) = (CField_{j,i} - simbeam_{j,i} - CP_{j,i}) x (e^{-(w12j - idwj,i)dt}) + CP_{j,i}$$

[2] M. Huening, " Selbstoptimierende Parametersteuerung der Hochfrequenz des Supraleitenden Linearbeschleunigers TESLA Test Facility", Master thesis, RWTH/DESY, June 1998

$$CP_{j,i}(MV) = (2w12_j/(w12_j - idw_{j,i}))xCDrive_{j,i}$$

At the klystron level the phase and amplitude of the RF signal are regulated using vector-sum regulation. Certain cavities also have a phase-shifter. Both feedback systems, together with the feed-forward set-table (Eq. 4-8) determine the *Vfwd* function. This function is calculated with Eq. (4-14), explained later. The vector-sum control is described in further detail later (section 4-8). The effect of the fast ferrite vector-modulator is calculated at a later step with another script, also described later in this document (section 4.9). The cavity voltage and the forward-power settings are integrated from time step to time step. The beam-loading function is discussed next. Also shown in Fig. 4.9 is that the voltage variation during each time step is less than 1%. This is a benchmark figure for what the feedback control systems have to be able to achieve.

Beam-Loading

The beam-loading voltage $\Delta Vbj,i$ in each cavity j is calculated in **SimulateField.m** for each time step i with Eq. (4-12). A derivation of this formula is given in appendix B. Given below is the same formula in the notation used in the program.

$$\Delta \boldsymbol{Vb}_{j,i} = \sum_{k} \left[\boldsymbol{Iini}_k \boldsymbol{T'}_{j,k} \, \boldsymbol{e}^{\left(-i\left(\boldsymbol{\phi}0_j - \boldsymbol{\omega RF}_j\left(\boldsymbol{tsync}_{j,k} - \boldsymbol{ta}_{j,k}\right)\right)\right)} \right]_i \boldsymbol{\omega RF}_j \boldsymbol{r}_j \Delta \boldsymbol{t} 10^{-6} \qquad (\boldsymbol{MV}) \qquad (4\text{-}12)$$

$$simbeam_{j,i}(MV) = IFac_{j,i}xCCur_{j,i} = \omega RF_j x r_j x \Delta t x 10^{-6} x CCur_{j,i} x e^{-i\phi 0j}$$

The phase-factor $(\omega RF(tsync_j - ta_{j,k}))_i$ is calculated from the arrival time of the macro-particle k calculated with **acceleration.c** on the basis of the cavity fields (which is calculated from beam loading and feedback in the previous time step). The macro-particle currents are weighted with the macro-particle phase-factors (which are relative to that of the synchronous phase). It obviously also includes the fixed beam phase advance defined in the input. The beam-loading also includes the normalized cavity shunt impedance, *r (=Rsh/Q0)*, the cavity frequency, ωRF, and the transit time factor, $T'_{j,k}$. $T'_{j,k}$ is a matrix which contains TTF of the macroparticle k in the cavity j (calculated with **acceleration.c**), taking into account the actual βb of each macro-particle in each cavity. The synchronous particle arrival time, $tsync_{j,k}$ is a matrix made from k vectors of j length, which contains the arrival

time for the synchronous particle – with the same column copied *k* times. Multiplication over the time step converts the bunch current *Ib* to a charge, which is the relevant parameter for beam-loading. As shown in Fig. 4-9 the beam loading voltage is subtracted from the actual voltage instantaneously, producing a step in the voltage function, since the beam passes through the cavity in a very small fraction of the time step. Also indicated in the equation above is that the output parameter *CCur* represents the beam-loading. Upon division by $T'_{j,k}$ (which is hard to calculate during post-processing but can be replaced by the synchronous transit-time factor as a first order approximation), *CCur* becomes the beam current.

Forward Power Feedback

Eq. 4-14 describes the *Vfwd* function. *SVfwd* is the forward voltage signal including vector-sum and vector-modulator feedback. At the core of this calculation is the calculation of the average difference voltage vector $\boldsymbol{Vcav}_j \boldsymbol{v}_{j,i} - \hat{\boldsymbol{V}}_{j,i}$, which is the basic signal for the vector-sum feedback. In fact, the difference signal between nominal field and measured field, $\left\{\left(\boldsymbol{Vcav}_j \boldsymbol{v}_{j,i} - \hat{\boldsymbol{V}}_{j,i}\right) \times \boldsymbol{G}_j\right\}_l$, also needs to be averaged over the module (using the **dimsum.m** routine) in the case of the vector-sum control. The effect of the vector-modulator is accounted for in the complex attenuation *Aps*. This complex factor is calculated in the **dophaseloop.m** routine. The individual cavity attenuation a_j is also multiplied to *SVfwd*.

The voltage one actually gets in the cavity for a voltage *SVfwd* coming down the coupler is reduced in the presence of detuning (un-matched cavity impedance) and coupling loss. As is explained in appendix B the forward voltage signal needs to be multiplied with an attenuation and a phase-factor, (Eq. 4-13), in this case. Note the asterisk to *Vfwd**, needed because the definitions of the forward wave voltage in Eq. (4-13) and (4-14) differ slightly. *Vfwd* in Eq. (4-14) will receive the factor in the parenthesis when inserted in Eq. (4-11). Strictly speaking the set voltage *SVfwd* is only reached in the cavity after a time $t > \tau f$ (in the ideally tuned case). The physics on which Eq. (4-13) is based is explained in appendix B.

$$\boldsymbol{Vfwd}^*_{j,i+1} = \frac{2\boldsymbol{\omega}12_j \times \boldsymbol{SVfwd}_{j,i}}{\left(\boldsymbol{\omega}12_j - \boldsymbol{i}\Delta\boldsymbol{\omega}_{j,i}\right)}\left(1 - \boldsymbol{e}^{-\left(\boldsymbol{\omega}12_j - \boldsymbol{i}\Delta\boldsymbol{\omega}_{j,i}\right)\Delta t}\right) \qquad (\boldsymbol{MV}) \qquad (4\text{-}13)$$

Also included in Eq. (4-14) is the reflection from the cavity to the vector-modulator (the second term in the parenthesis in the first line of Eq. 4-14). The reflection arises from the attempt of the vector-modulator to change the field in the cavity by the amount $\left(\hat{V}_{j,i} - \boldsymbol{SVfwd}_{j,i}\right)$. The reactive power *Rps*, which is calculated also in **dophasellop.m**, is the reflected power related to the sinus of (half of) the phase-difference between the reflected signals returning into the two branches of the phase-shifter. This reflection is sent back to the cavity, where reflection happens again. Given the many RF periods within a time step this term becomes a converging geometrical series *Rps*/(1-*Rps*). This behavior would not occur if the phase-shifter had an additional circulator between phase-shifter and cavity.

Eq. (4-14) summarizes the forward voltage calculation, where $\boldsymbol{Vcav}_j\boldsymbol{v}_{j,i}$ is the set-field in the cavities. The exponential factor from Eq. (4-13) will be added in the program to *Vfwd* in Eq. (4-11). The normalized fill function is included here because the vector-sum control is also active during the filling. $\sqrt{2\boldsymbol{QL}_j\boldsymbol{r}_j}$ is the conversion factor, converting *Sfwd* into a voltage (*SVfwd*). More details on the vector-sum contribution can be found in section 4.8. More details on the phase-shifter contribution can be found in section 4.9.

$$\boldsymbol{SVfwd}_{j,i} = \left\{ \left[\frac{\left(\boldsymbol{Vcav}_j\boldsymbol{v}_{j,i} - \hat{V}_{j,i}\right)\times \boldsymbol{G}_j}{\sqrt{2\boldsymbol{QL}_j\boldsymbol{r}_j}\times \boldsymbol{a}_j} + \boldsymbol{Sfwd}0_{j,i} \right] \times \sqrt{2\boldsymbol{QL}_j\boldsymbol{r}_j} \times \boldsymbol{Aps}_{j,i} \times \boldsymbol{a}_j \right\}_l$$

$$\boldsymbol{Vfwd}_{j,i+1} = \frac{2\boldsymbol{\omega}12_j}{\left(\boldsymbol{\omega}12_j - \boldsymbol{i}\Delta\boldsymbol{\omega}_{j,i}\right)} \left[\boldsymbol{SVfwd}_{j,i} + \frac{\boldsymbol{Rps}_{j,i}}{\left(1-\boldsymbol{Rps}_{j,i}\right)}\left(\hat{V}_{j,i} - \boldsymbol{SVfwd}_{j,i}\right) \right] \quad (\boldsymbol{MV}) \tag{4-14}$$

The output parameters *CDrive* and *CForw*d contain the relevant components of the expressions above (components in square brackets) and can therefore be used to reconstruct the control signals and forward power.

$$SVfwd_{j,i} = CForwd_{j,i}(MV/(\Omega)^{1/2}) \; x \; (Aps_{j,} \; x \; a_j \; x \; (2QL_jr_j)^{1/2}$$

$$Vfwd_{j,i} = 2w12_j/(w12_j - idw_{j,i}) \; x \; CDrive_{j,i}(MV)$$

The forward power (in kW) that the klystron needs to supply can be calculated from $(CForwd^2) \times 10^9$. The actual forward wave voltage change happening in the cavity at each time is:

$$\Delta Vfwd_{j,i}(MV) = (2 \times Vfwd_{j,I} \times e^{-(w12j - i\Delta wj,i)}) / (1 - \Delta w_j / w12_j) \ .$$

Note that one can calculate the forward power actually delivered from the klystron with $(CForwd^2) \times 10^9$ because of the presence of feedback. In reality the attenuation due to detuning and coupling loss would be pre-compensated by the klystron, by increasing the output power by the respective factor (and this would improve the feed-forward component). This step was not explicitly included in S.C.R.E.A.M. Note, however, that the feedback system will notice during the next time step that less power than needed has actually reached the cavity because of detuning and coupling loss (see Eq. 4-13 for the voltage reduction factor). Therefore the feedback signal will be bigger by the respective amount in the next time step. Note that *CForwd* also includes the gain factor of the vector-sum control. Similarly as with the attenuation due to detuning and coupling loss the feedback circuit automatically regulates the input power and voltage to the desired level and therefore corrects for the gain factor by reducing the control signal before amplification. If the control system works properly the power provided by the klystron after filling should match the power removed by the beam (for zero detuning). This is a “sanity-check” for the forward power signal.

The **SimulateField.m** calls **dophaseloop.m** after completing the calculation of (Eq. 4-14).

Miscellaneous

In the context of the TESLA R&D, fast (Piezo-) and slow (blade-) tuners were developed that successfully compensated for microphonics and Lorentz-force detuning in the TTF linac. These are NOT implemented in this program. The program assumes that the slow tuner successfully compensates for the cavity detuning during cool-down. The Lorentz-force detuning compensation just consists in guessing the total detuning at full field and feed-forward correcting for it with a constant pre-detuning of the cavity. A Piezo-tuner could be easily simulated in the program by reducing the actual Lorentz-force detuning constants (*Cavities.KLore*ntz).

Save

As the final procedure **SimulateField.m** reduces the number of columns (=number of bunches) of all fields in the *cr* structure by the *Downsample* parameter from the General field (i.e. only every *Downsample*th time step is saved). Since all cavity properties, such as forward power and feedback, detuning, etc,.. evolve slower than the time-step the calculation of the full array is necessary. The down-sampling can only be done after the fact. *NrunxNfiles* is the total number of pulses simulated. The loops over *Nrun* and *Nfiles* are implemented in **scream.m**. The procedures used to reduce the number of data after *NrunxNfiles* runs are discussed there.

4.7 detuning.m

The detuning routine calculates the cavity Lorentz-force detuning in each cavity at every time step from the cavity fields in the preceding time step. It is called for each time steps of an RF pulse by **SimulateField.m**. In the function call it receives the accelerating voltage in the cavities of the preceding time step ($\hat{V}_{j,i-1}$), the frequency change due to detuning in the preceding time step ($\Delta\omega_{j,i}$), the time step (Δt) and the current Lorentz-detuning constant of each cavity (KL_j). The Lorentzforce detuning constants for each cavity, KL_j, are determined once in the input, and varied randomly with the relative Lorentz-detuning factor, ΔKL_j, once every file (in **scream.m)**. The Lorentzforce detuning is not believed to change once the accelerator is built. The *KL* constants are therefore fixed for all "runs" within a "file" (more in the discussion of **scream.m**). It returns the frequency change due to Lorentzforce detuning $\Delta\omega_{j,i}$ in each cavity at timestep *i*. The output field *cr.dw* contains the total detuning (as given in (Eq. 4-4)).

The mechanical rigidity of the cavity delays the change of frequency in the cavity. Therefore detuning of the cavity is characterized by the time constant τc. Although the system has many mechanical resonance frequencies, the process is described well enough with only one time constant. This time constant is defined in the **detuning.m** routine and was chosen to be 300 μs [3]. Eq. (4-15) gives the formula used to describe the frequency change of the cavity due to Lorentz-force detuning:

[3] V. Ayvazyan, S. Simrock, "Dynamic Lorentz Force Detuning Studies in TESLA Cavities", presented at the European Particle Accelerator Conference 2004, Lucerne, Switzerland, July 2004

$$\Delta\omega LF_{j,i} = \Delta\omega LF_{j,i-1}\left(1-\frac{\Delta t}{\tau c}\right)+2\pi\frac{\Delta t}{\tau c}KL_j\left(\hat{V}_{j,i}\right)^2 \quad , \tag{4-15}$$

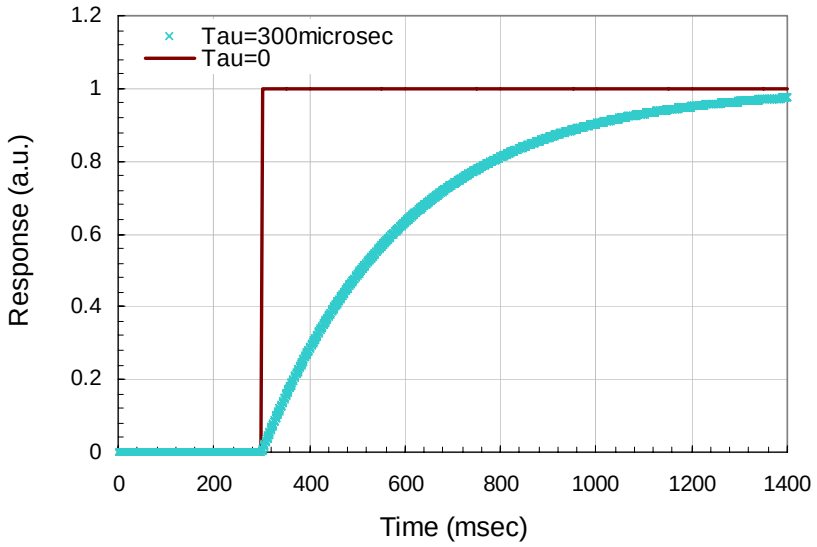


Figure 4-10: Response function to step calculated with Eq. (4-15).

where the $\Delta t/\tau c$ factor dampens the frequency response very much like a low-pass filter. When a sudden electric field step is applied to the cavity the detuning first is negligible (a $\Delta t/\tau c$ fraction of the prescribed *KL*). After a time $t \sim \tau c$, however, the detuning converges to the prescribed (DC) value. Fig. 4-10 shows the response function calculated with Eq. (4-15) for a step-function excitation to an arbitrary DC amplitude of 1. The approach function is exponential.

The Lorentz-detuning factor for each cavity KL_j is re-calculated for every "*file*" in **scream.m** with the help of the MATLAB *randn* function (Eq. 4-16).

$$KL_{j,new} = KL_{j,old}\left(1+\Delta kL_j R_j\right) \quad \left(\frac{Hz}{(MV)^2}\right) \tag{4-16}$$

The initial values of KL_j are those given in the input (*Cavities.KLorentz*). **Detuning.m** also includes a provision for the case in which no *KL* is defined in the input. KL_j=-1Hz/(MV)^2 is assumed in this case.

4.8 dimsum.c

At the klystron level the phase and amplitude of the RF signal is regulated using vector-sum regulation. The vector-sum regulation consists in summing the measured complex field vectors from all cavities within an RF unit (=driven by one klystron), measure its amplitude and phase with respect to some set-value and derive a control signal that will, as best as possible, drive the cavity field to the set phase and amplitude. Fig. 4-11 shows a schematic of the DESY/TTF vector sum control module. The phase and amplitude measurement is done after mixing with a signal with a slightly different frequency. The measurements can then be done on the lower beat frequency signal. S.C.R.E.A.M assumes that the feedback delay is 1 μsec. This is actually optimistic – the TTF DSP system has a total processing time of ~4 μsec. Future, FPGA based systems promise to reach the 1 μsec mark, however.

The actual field amplitude is usually complex and thus includes information about the cavity detuning (cavity phase-factor). The program assumes that the RF system phase is zero and therefore the correction aims at restoring zero phase. In a real (proton) linac the synchronous phase has to be determined for every cavity separately using beam-loading. The so measured values (which also include cable delays) are stored in the set-table and subtracted from the phase-signal. The set-value in S.C.R.E.A.M, *Vcav,* is real, so the phase factor

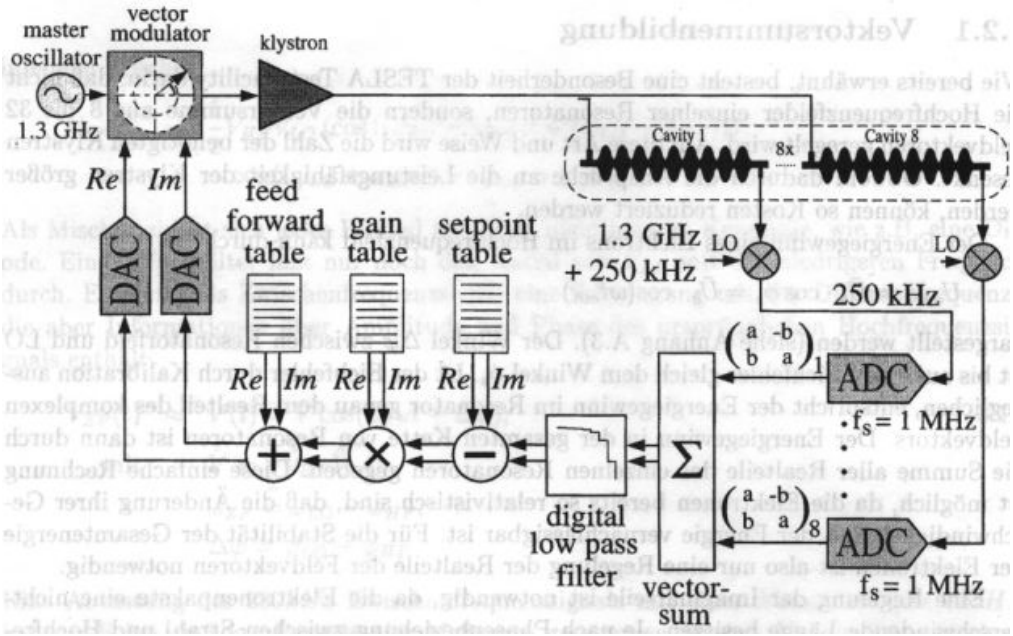


Figure 4-11: Schematic of vector-sum control in DESY/TTF.

of the difference contains the entire cavity phase shift as a result of detuning. The difference signal is redirected toward the input after multiplication with a proportional gain *G* and a cavity attenuation factor *a*, both defined in the input. The vector-sum control process is described entirely with Eq. (4-14), where *VFwd* is the forward voltage, and $Vcav_j v_{j,i}$ is the set-field in the cavities. Note that the actual power that needs to be supplied to achieve a desired voltage is larger by a certain factor (see first line in Eq. (4-14) to take into account the fact that the cavity is detuned (as well as coupling loss). The derivation of these factors is given in appendix B. As discussed before in the context of the forward power, the feedback system automatically takes care of this issue.

Since vector-sum regulation is done at the klystron level, all the relevant parameters (gain, attenuation, difference signal) need to be averaged over the RF module. **SimulateField.m** uses **dimsum.c** to sum the difference signal over all cavities in each module and to determine the optimal feedback response at the klystron level, the crucial step in vector-sum control. The **dimsum.c** routine performs partial sums of elements of a (complex) vector passed on to it through the function call. The routine is optimized for fast processing (and therefore uses C language).

In particular, **dimsum.c** is called once every time step *i* by **SimulateField.m** to sum the amplified difference between (real) set-field, *Vcav* and the actual, complex accelerating voltage $\hat{V}$ in each cavity *j* belonging to the module *l*. The voltage difference is divided by the conversion factor $(2QLr)^{1/2}$ to convert it to the square root of power. It is multiplied by the feedback gain and divided by the cavity attenuation. The routine uses a simplification when applying an average of the (proportional) gain of all cavities in each module to amplify the difference between the set and the actual field. The real and imaginary parts of $\hat{V}$, which are passed on to **dimsum.c** in the function call from **SimulateField.m**, are summed separately. The most time consuming part of the routine is related to defining which cavities belong to a module. To that end the function call also includes the *Module* vector from the *Cavities* structure (see discussion of input) and the *Nmod* (= number of modules) variable. *Cavities.Module* is a size *Ncav* vector that contains the module number to which each cavity belongs. This vector is renamed as idx_j in **dimsum.c**. The components of this vector are used as the index of a temporary vector, *SP*, which sums all the amplified difference-signals *j* into the element *l*, where *l* is the module number. This *l*-size vector is then augmented

back to size *j*, with all cavities in one module having the same difference signal value, the average per module.

4.9 initphaseloop.m and dophaseloop.m

Dophaseloop.m describes a possible implementation of the fast ferrite vector-modulator with the characteristic parameters as defined in *Phaseloop*. The following describes an implementation of phase-shifters based on an approximation of a proportional-differential (PID) regulator. Any other design is also possible (first trials with a time-optimal regulator were also done but are not discussed further here). A disadvantage of the PID regulator is its sensitivity to noise. **Dophaseloop.m** requires prior initialization via **initphaseloop.m**. The initialization script is called by **SimulateField.m**.

The fast ferrite vector-modulator is made out of two phase-shifters, which each act independently on one half of the forward power. Fig. 4-12 shows a schematic of a possible implementation of the vector-modulator. It consists of two ferrite loaded stubs with a bias coil for each. The phase-shift in each branch is set via its bias-field. Eq. (4-17) describes the effect of the vector-modulator on the forward power. If the two branches produce the same phase-shift ($\psi 1 = \psi 2$), the attenuation is 1 (the reflection is zero) and the effect consists only of a phase-shift $(\psi 1 + \psi 2)/2$. If the phase-shifts in the two branches are not the same, reflection occurs, and the forward signal is attenuated with by $\cos((\psi 1 - \psi 2)/2)$. In other words the phase-shift, ϕps, depends on the average phase and the attenuation A depends on the phase-difference. Note that (4-17) also includes $\psi 0$, the initial phase-shift before the effect of the biased ferrite (e.g. provided by an extra-length of wave-guide). This initial phase-shift allows the phase-shifter to operate at a different point than zero (at the expense of additional signal attenuation, however). The initial phase-shift is assumed to be symmetrically distributed ($\psi 1 \rightarrow \psi 1 + \psi 0$, $\psi 2 \rightarrow \psi 2 - \psi 0$) such as to not cause any residual phase-shift.

$$\begin{aligned} Aps &= \frac{Pfwd}{Pin} = e^{i\left(\frac{1}{2}(\psi_1+\psi_2)\right)} \times \cos\left(\frac{1}{2}(\psi_1-\psi_2)+\psi_0\right) \\ Rps &= \frac{\mathrm{Pr}\,eact}{Pin} = e^{i\left(\frac{1}{2}(\psi_1+\psi_2)\right)} \times \sin\left(\frac{1}{2}(\psi_1-\psi_2)+\psi_0\right) \end{aligned} \qquad (4\text{-}17)$$

The model implementation of the vector-modulator in the **dophaseloop.m** script is strongly simplified (see block-diagram in Fig.

4-12). The elements contained are: -1- the PID regulator, -2- a saturation condition reflecting the limited “range” of the phase-shifter and –3- a (low-pass) filter element that takes into account the finite reaction time of the phase-shifter (mainly due to the inductance of the bias coils and the voltage limitation of the bias coil power-supplies).

After initialization, **dophaseloop.m** calculates the correction signal, Eq. 4-18. The to be corrected error signal $\Delta\hat{V}ps_{g,i}$ is the difference between the set field and the actual field in each cavity *g* (*g* because only the cavities *g* have a vector-modulator) after subtraction of the correction provided by vector-sum control. Therefore the second term in Eq. 4-18 is the average difference signal over the module *I*. This prevents the phase-shifters from working against the vector-sum control. The modulus of $\Delta\hat{V}ps_{g,i}$ is the amplitude error and the imaginary phase angle the phase error.

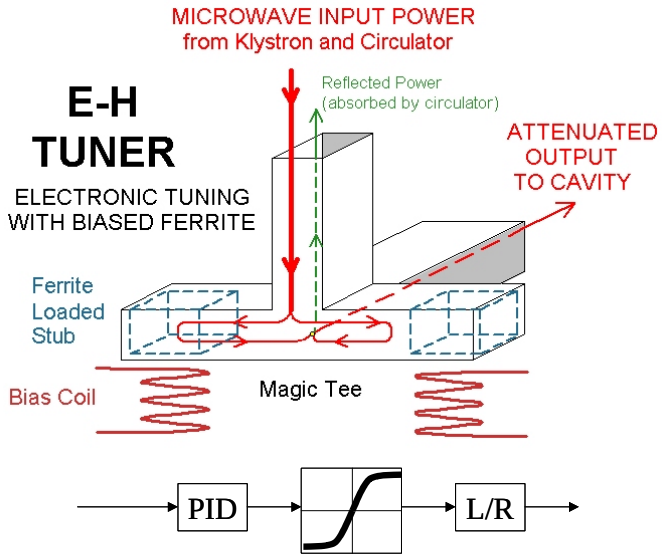


Figure 4-12: Schematic of fast ferrite vector-modulators for the Fermilab PD (wave-guide type).

$$\Delta\hat{V}ps_{g,i} = \left[\left(Vcav_g v_{g,i} - \hat{V}_{g,i}\right) - \overline{\left(Vcav_g v_{g,i} - \hat{V}_{g,i}\right)_I}\right] \quad (MV) \qquad (4\text{-}18)$$

Below is a step-by-step discussion of the effect of the different phase-shifter components listed above on the correction signal.

The two signals *S1* and *S2* refer to the amplitude and the phase correction. Eq. 4-19 describes the PID regulator, with the proportional (first term) and differential (second term) components. The weighting

between the components does not vary from before to after filling. Usually the differential component is chosen to be much stronger than the proportional gain. This ensures that the phase-shifter reacts faster. The respective input parameters in *Phaseloop* are AmpGain (*GAps*) and AmpDGain (*DAps*) for the amplitudes (S1) Gain (*Gpps*) and DGain (*Dpps*) for the phases (S2). Eq. (4-19a) describes signal 1 (amplitude) and (Eq. 4-19b) the phase signal. Note that Eq. (4-19) is slightly inaccurate – it should in fact calculate the amplitude and phase from the modulus and the phase of the complex control signal. At a small phase, however, the real and imaginary parts do the job as well (because $\phi \sim sin\phi$), and faster. Fig. 4-13 shows the PID response function to a control signal $\phi ps = 10°$, $A = 1$. The proportional part is negligible against the differential part (spike) for the set of gains chosen.

$$S1_{g,i} = \Re\left(\Delta\hat{V}ps_{g,i}\right)GAps_g + \Re\left(\Delta\hat{V}ps_{g,i} - \Delta\hat{V}ps_{g,i-1}\right)DAps_g \quad (rel) \tag{4-19a}$$

$$S2_{g,i} = \Im\left(\Delta\hat{V}ps_{g,i}\right)Gpps_g + \Im\left(\Delta\hat{V}ps_{g,i} - \Delta\hat{V}ps_{g,i-1}\right)Dpps \quad (rad) \tag{4-19b}$$

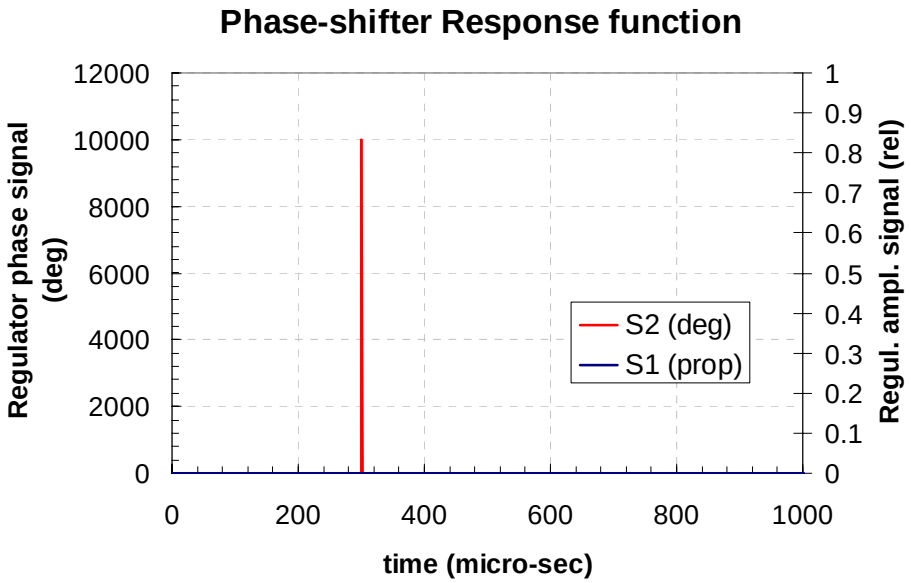


Figure 4-13: PID-signal as response to a 10° phase signal on the vector-modulator input. No amplitude attenuation signal (S1=0). (GAps=0.04, DAps=200, Gpps=0.2, Dpps=1000)

Eqs. (4-20 & 4-21) describe the calculation of the phase-shifts required in each branch, PS1 ($\psi 1$) and PS2 ($\psi 2$) to deliver the amplitude and phase correction. Here *PS1* and *PS2* stand for the phase-shifts (in rad) provided by the two different branches of the vector-modulator. Each branch takes care of part of the requested

phase-shift and attenuation. The amount of phase-shift needed in each branch of the phase-shifter for a given total phase shift, ϕps, and attenuation, Aps, can easily be calculated:

$$\psi 1 = \phi ps + arccos(Aps),\ \psi 2 = \phi ps - arccos(Aps),$$

ϕps, the total phase-shift is $(1/2(\psi 1 + \psi 2))$ and *Aps* the (relative) amplitude is $cos(1/2(\psi 1 - \psi 2))$. That does not include the effect of the working point $\psi 0$. The best working point (because of fastest slew rate) is $\psi 0 = \pm 45°$ in branches 1 and 2 (albeit reducing the transmitted power by one half). No net phase-shift is therefore added as a result of $\psi 0$. Eq.(4-20) implements the above, including the working point $\psi 0$. Eq. (4-20) also includes a saturation condition. When the angle becomes larger than ψsat, the phase-shifts P*S1 or PS2* cannot follow. This is the result of the use of the arctan function, which saturates at 90° when the argument diverges.

It also needs to be noted that Eq. (4-20) mixes the attenuation and phase signals *S1* and *S2*. If $\psi 0 = 0°$ the amplitude signal is some value close to zero (and not close to one as is *Aps*). As with the vector-sum control feedback discussed in 4.8, this is allowable since Eq. (4-20) in fact is the part of a feedback loop, which auto-corrects for "wrong" start values. The mixing of the feedback loop and the real physics description can give rise to confusion. In the case in which $\psi 0 \neq 0°$ the amplitude working point shifts from zero to $\psi 0$ *(+S1).* This is taken into account with the $1/sin(\psi 0)$ factor.

$$\mathbf{PS}1_{g,i} = 2\psi sat/\pi \cdot \mathbf{arctan}\left[\frac{\mathbf{S}2_{g,i} + \frac{\mathbf{S}1_{g,i}}{\mathbf{sin}(\psi 0)}}{2\psi sat/\pi}\right] \quad (\mathbf{rad})$$

$$\mathbf{PS}2_{g,i} = 2\psi sat/\pi \cdot \mathbf{arctan}\left[\frac{\mathbf{S}2_{g,i} - \frac{\mathbf{S}1_{g,i}}{\mathbf{sin}(\psi 0)}}{2\psi sat/\pi}\right] \quad (\mathbf{rad}) \qquad \text{(4-20a/b)}$$

Fig. 4-14 shows an exemplary implementation of Eq. (4-20) Fig. 4-15 shows the effect of the arctan saturation function in Eq. (4-20).

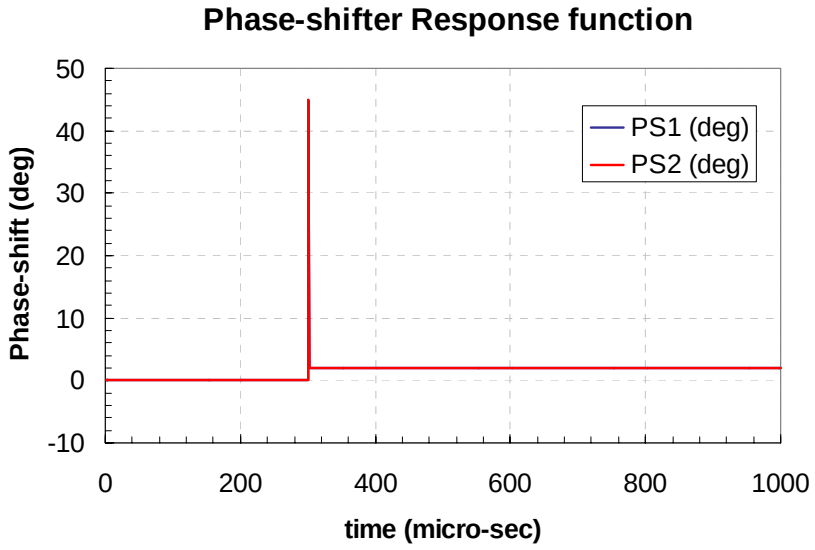


Figure 4-14: Saturator response to a 10° phase signal after transformation by the PID. Obviously ψsat is 45° in this example. No amplitude attenuation signal (S1=0). (GAps=0.04, DAps=200, Gpps=0.2, Dpps=1000)

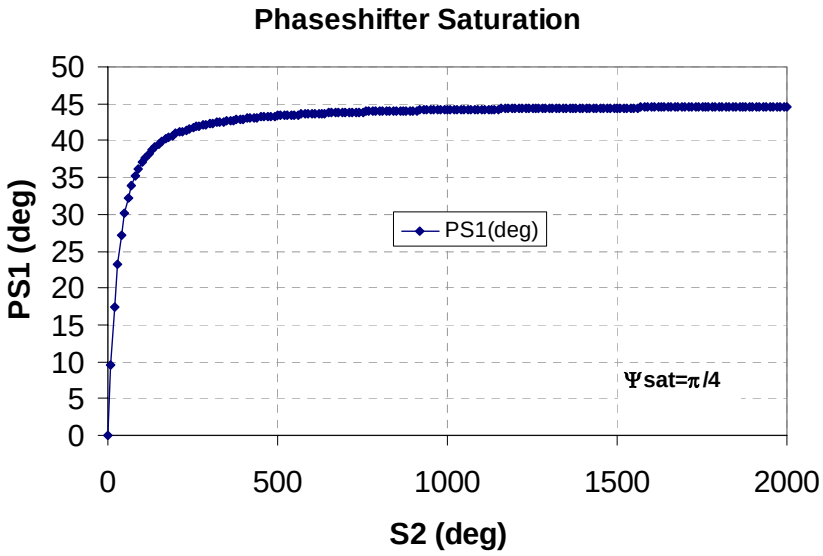


Figure 4-15: Saturation function as implemented in Eq. (4-20).

Eq. (4-21) describes the low pass filter element that implements the finite reaction time of the phase-shifter (as a result of the solenoid inductance and the power supply limitations). The phase-shifter signal is integrated and the actual signal only provides an increment of timestep/τps to the total signal. τps, which is given in *General.PhaseTau*, is the phase-shifter time constant. PS1' and PS2' are also similar to the output phase-shifts *sh1* and *sh2* of the two phase-shifters (in rad).

$$PS1'_{g,i} = PS1'_{g,i-1} + PS1_{g,i}\frac{\Delta t}{\tau ps} = sh1_{g,i+1} \quad (rad)$$
$$PS2'_{g,i} = PS2'_{g,i-1} + PS2_{g,i}\frac{\Delta t}{\tau ps} = sh2_{g,i+1} \quad (rad) \qquad (4\text{-}21)$$

Fig. 4-16 shows the slowed response of the vector-modulator as a result of the *L/R* time constant as implemented in Eq. (4-21).

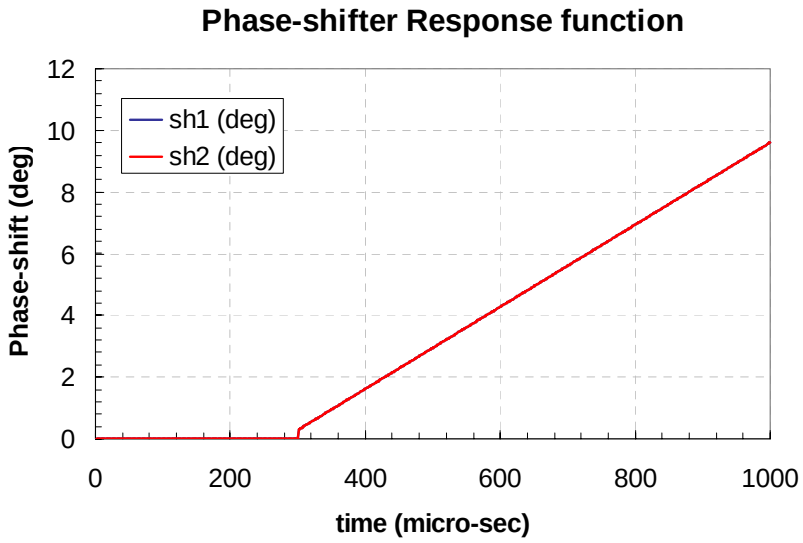


Figure 4-16: Effect of the L/R time constant on the response of the vector-modulator to a 10° phase-shift signal. (τps=150 μsec).

The main **dophaseloop.m** output, however, is *CFwdpl* (or *Aps*) the complex factors (relative to 1) with which the forward power from the klystron needs to be multiplied to simulate the phase-shifter attenuation (and phase-shift). Those factors are calculated with Eq. (4-22).

$$Aps_{g,i+1} = \frac{1}{2\cos(\psi 0)}\left[e^{i(sh1_{g,i}-\psi 0)} + e^{i(sh2_{g,i}+\psi 0)}\right] \quad (rel) \qquad (4\text{-}22)$$

$$Rps_{g,i+1} = GR_g \cdot \sin\left(PS1'_{g,i}\right)e^{i\left(PS2'_{g,1}\right)} \quad (rel) \qquad (4\text{-}23)$$

Eq. (4-22) "cheats" in the sense that the attenuation due to operation at $\pm\psi 0$ is removed from the reported attenuation. Note that,

apart from the removal of $\psi 0$, Eq. (4-22) corresponds exactly to Eq. (4-17), but written in a more "intuitive" way (the two exponential factors describe exactly what the two phase-shifter branches actually do!). The complex *Aps* and *Rps* are returned to **SimulateField.m** and used in Eq. (4-14). Note that if $\psi 0 \neq 0$ the modulus of *CFwdpl* can be larger than 1. In this case the increase of power from the phase-shifter comes from a reduction of the offset power reflection due to the chosen working point.

Fig. 4-17 shows the phase-shifter response calculated with Eqs. (4-19) - (4-22) for a step function control signal (demanding 10 deg at t=300 μsec). S1 was set to zero in this calculation, thus no attenuation was included in the control signal. The jump at 300 μsec is related to the strong differential gain (here 1000). It takes the phase-shifter ~650 μsec to deliver the requested phase-shift (the time constant assumed was 150 μsec). The strong differential gain does partly overcome the time-constant limitation of the phase-shifter. The signal attenuation shown is ~1% (-0.1dB), very close to zero. It is a result from moving the phase away from the working point. As mentioned above the attenuation due to the π/4 working point is not taken into account (this would add a constant offset of –3dB to the amplitude signal in Fig. 4-17). Besides the saturation of the control signal using the *arctan* function, there is no further limitation, such as for instance on the voltage of the power supply for the phase-shifter bias magnets. As mentioned before the as implemented vector-modulator does not include a circulator between it and the cavity.

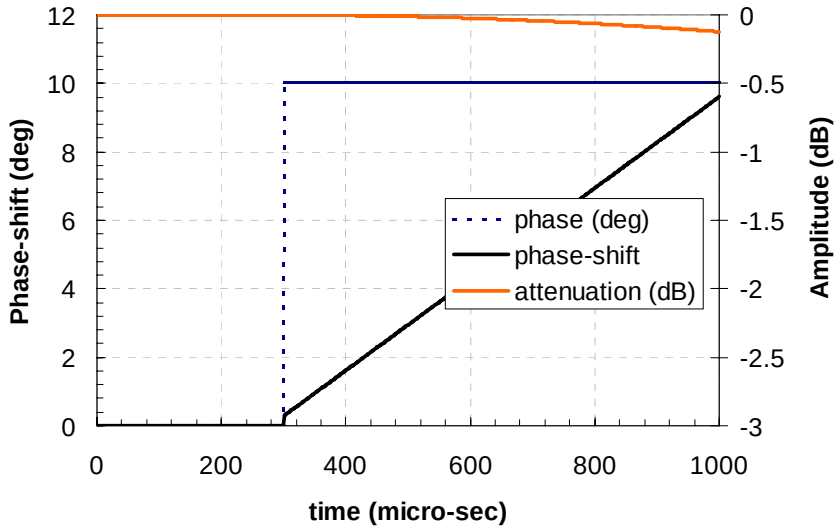


Figure 4-17: PID-based vector-modulator response as calculated with dophaseloop.m.

5 EXAMPLE – FERMILAB PROTON DRIVER

5.1 Fermilab Proton Driver

The proton-driver (PD) is a proposed accelerator at Fermilab that provides high power H^- beam at 8 GeV. Although the proton driver is a multi-mission machine its primary purpose is to produce neutrinos. Its other potential uses include the proton and antiproton production for the Tevatron and the Fermilab fixed target program as well as an X-ray FEL, an injector into a linear collider or a neutrino factory (muon collider) and a spallation neutron source. Besides very large beam power (2 MW @ 8 GeV), its multi-mission purpose, are the primary features of the proton driver.

The following will briefly discuss the result of a simulation of the proton driver to document the capabilities of the S.C.R.E.A.M code. Fig. 5-1 shows the layout of the proton driver linac as implemented in the program. Only the superconducting section (E= 87-8000 MeV) was

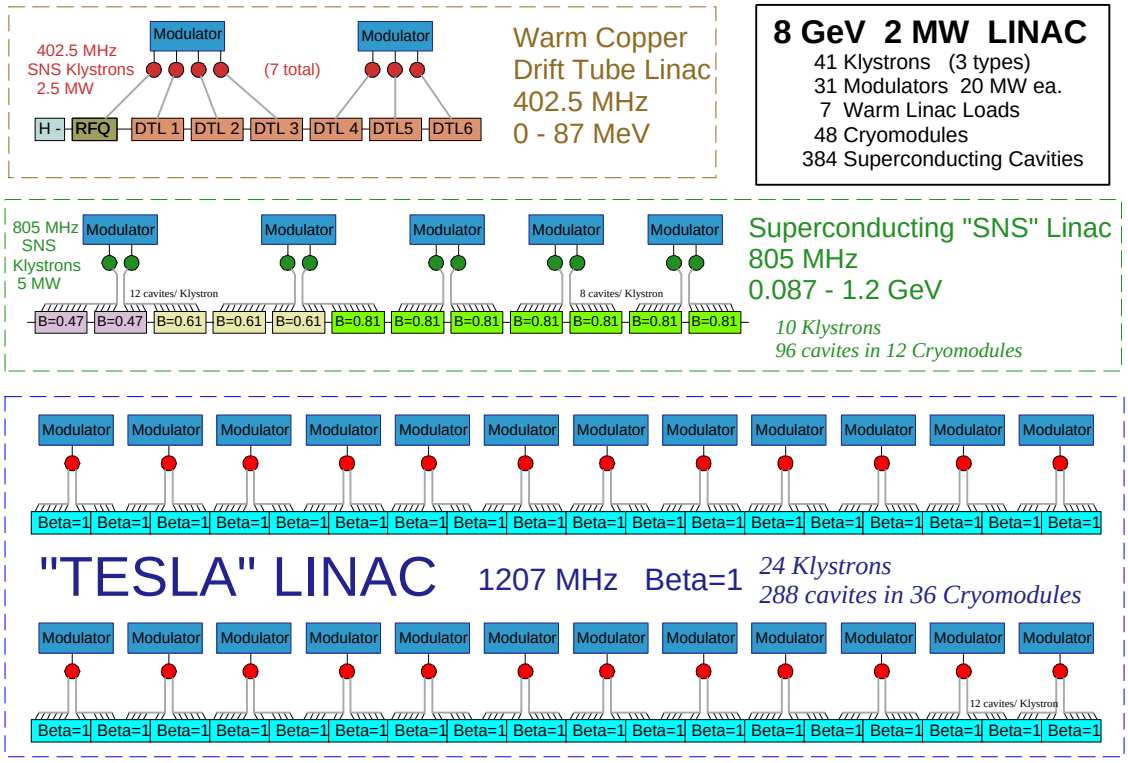


Figure 5-1: Layout of the proposed Fermilab proton driver (2 MW @ 8 GeV baseline) as simulated with S.C.R.E.A.M. below (G.W. Foster, 2003 Proton Driver Design study - from **http://tdserver1.fnal.gov/project/8GeVLinac/DesignStudy/**).

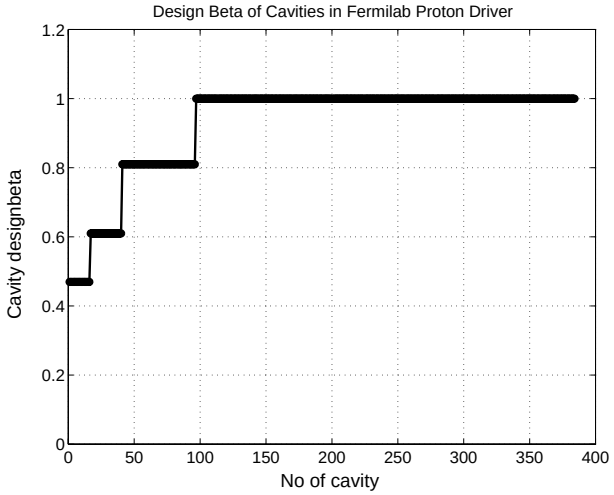
implemented in S.C.R.E.A.M. The linac consists of two β_c=0.47 cryo-modules, three cryo-modules with β_c =0.61 cavities, seven cryo-modules with β_c=0.81 cavities and 36 cryo-modules with β_c=1 cavities. All β_c<1 cavities are elliptical with six cells, very much like those proposed for the RIA accelerator by MSU. The β_c=1 cavities are elliptical nine-cell cavities, very much like those developed for the TESLA program. All cryo-modules hold eight cavities. There are eleven klystrons (each counted as a "RF-module" in the program) in the β_c<1 section and 24 RF modules (a one klystron each) in the β_c=1 section. The program therefore includes 35 modules. The β_c<1 section operates at the SNS frequency (805 MHz), the β_c=1 section at a "TESLA-like" frequency (1207 MHz). The total cavity count is 384. This particular PD design is one of two options currently under consideration.

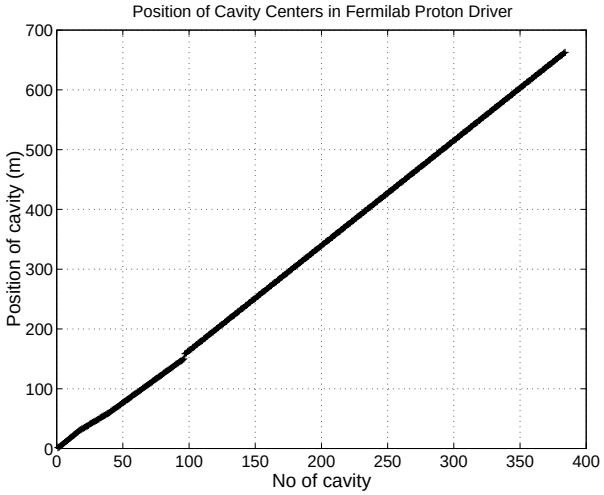
5.2 S.C.R.E.A.M input

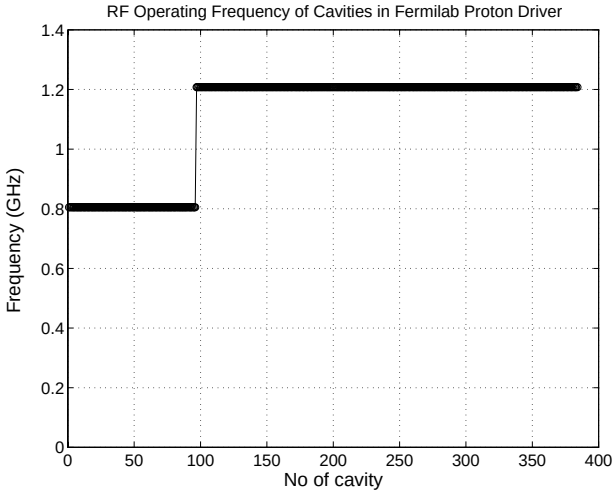
The following plots represent the major input parameters. The first set of plots describes the components of the linac (as shown in Fig. 5-1). Table 5-1 contains the main scalar parameters for the simulation discussed here. For details consult chapter 3.

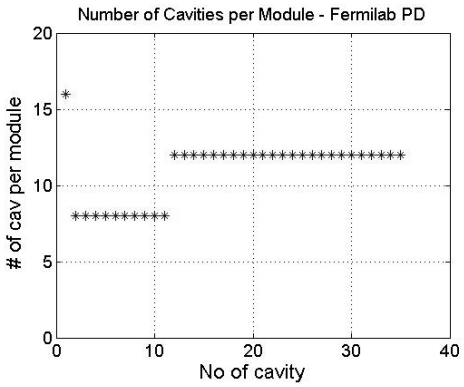
Table 5-1: Constant input parameters for S.C.R.E.A.M Proton-Driver simulation.

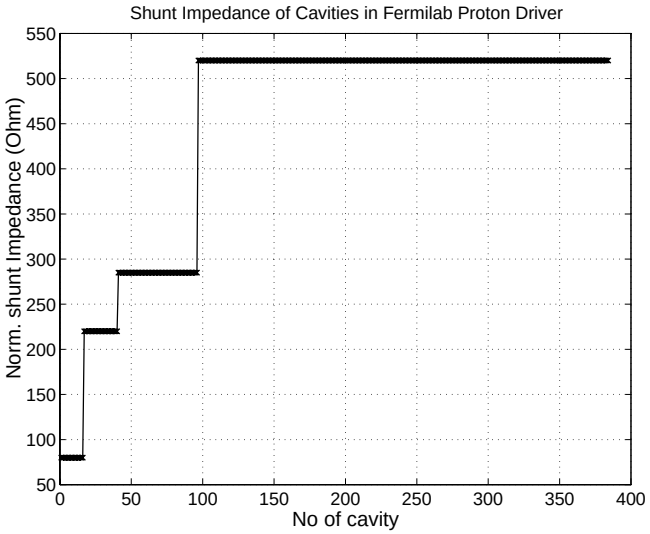
Field	Value
Nfiles	1
Nruns	1
doPhaseloop	1 (true)
Stepsize	1 μs
Downsample	1
Filltime	500 μs
Beamtime	800 μs
PhaseTau	150 μs
Ncav	384
Nm	35
Ng	96
Mode	π
GapLambda	0.5

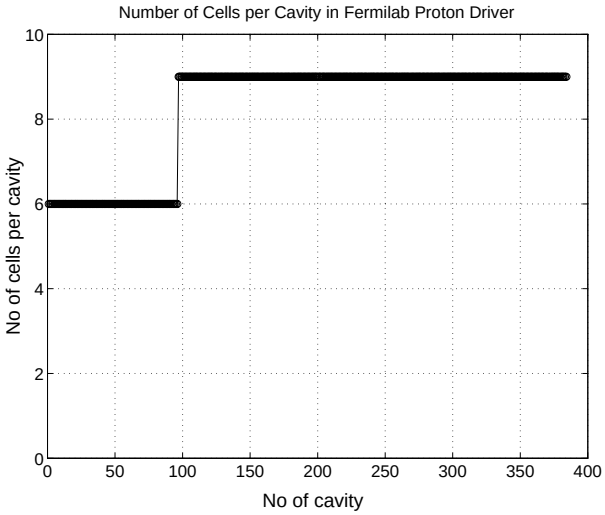

Design Beta of Cavities in Fermilab Proton Driver
Cavity designbeta
No of cavity

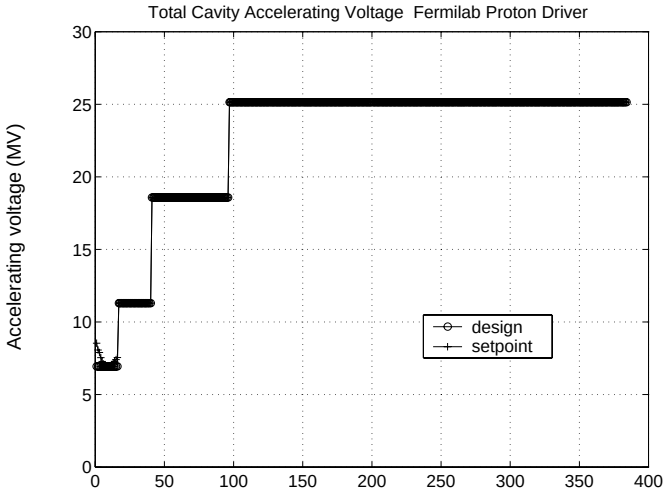

Position of Cavity Centers in Fermilab Proton Driver
Position of cavity (m)
No of cavity

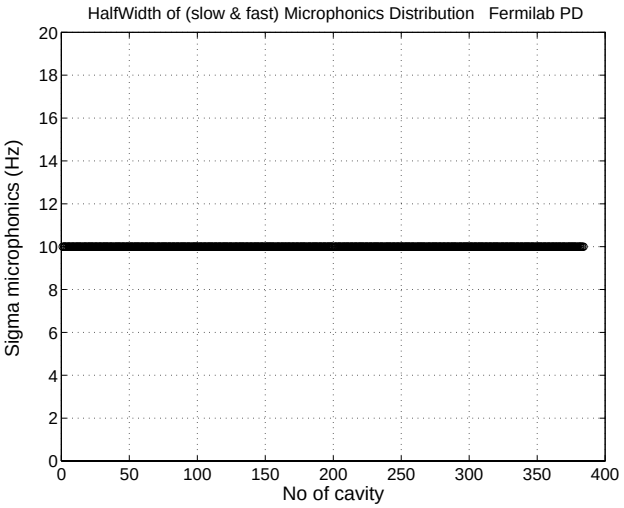

RF Operating Frequency of Cavities in Fermilab Proton Driver
Frequency (GHz)
No of cavity

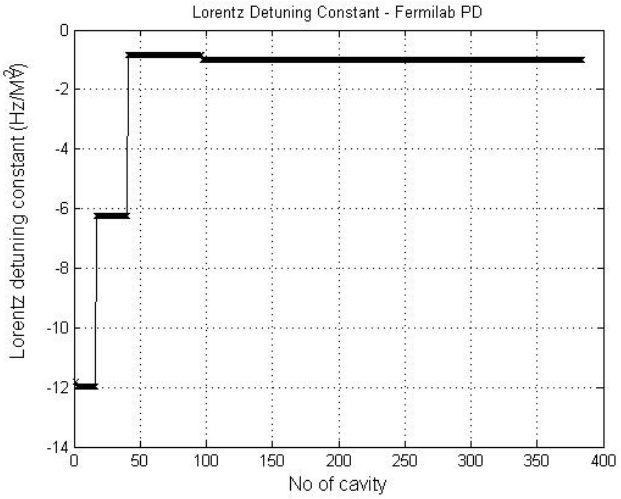

Number of Cavities per Module - Fermilab PD
# of cav per module
No of cavity

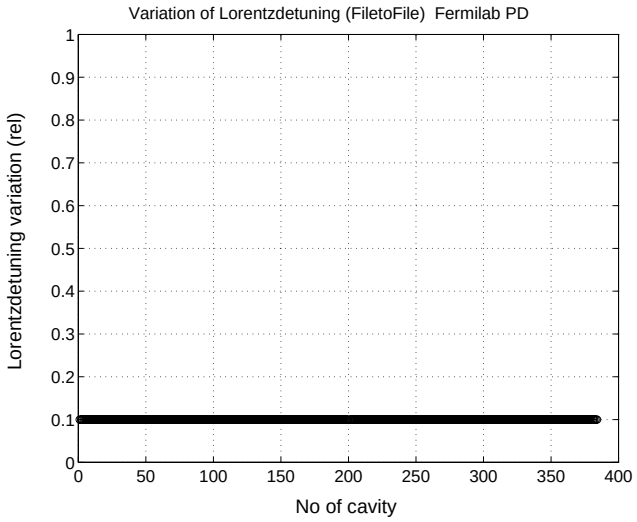

Shunt Impedance of Cavities in Fermilab Proton Driver
Norm. shunt Impedance (Ohm)
No of cavity

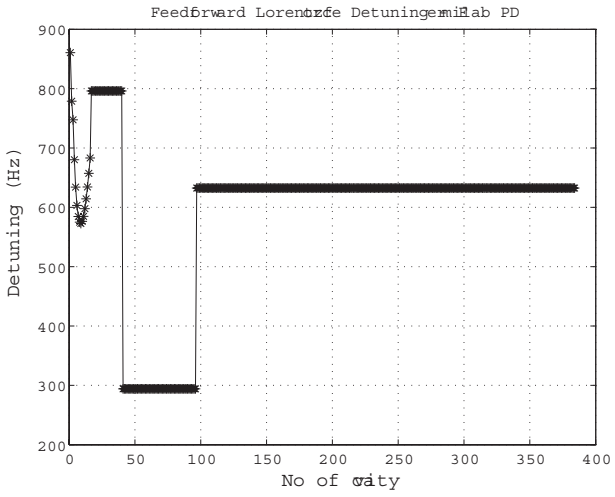

Number of Cells per Cavity in Fermilab Proton Driver
No of cells per cavity
No of cavity

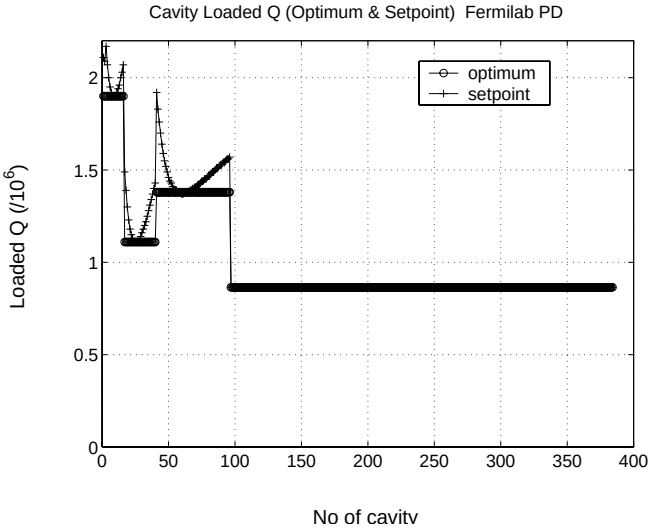

Total Cavity Accelerating Voltage Fermilab Proton Driver
Accelerating voltage (MV)
design
setpoint

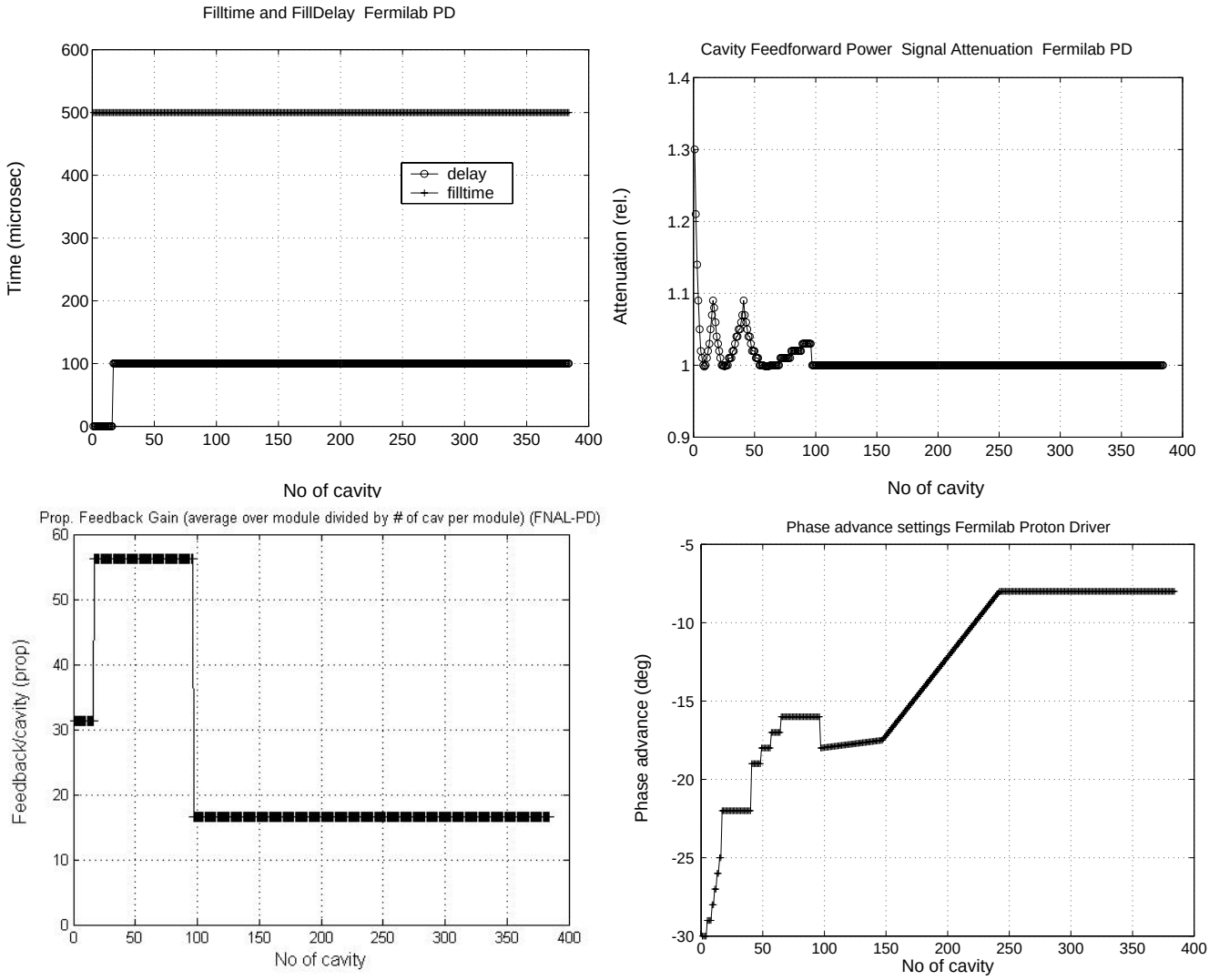

HalfWidth of (slow & fast) Microphonics Distribution Fermilab PD
Sigma microphonics (Hz)
No of cavity


Lorentz Detuning Constant - Fermilab PD
Lorentz detuning constant (Hz/MV²)
No of cavity


Variation of Lorentzdetuning (FiletoFile) Fermilab PD
Lorentzdetuning variation (rel)
No of cavity


Detuning (Hz)
No of cavity


Cavity Loaded Q (Optimum & Setpoint) Fermilab PD
optimum
setpoint
Loaded Q (/10⁶)
No of cavity



The optimum loaded Q (*QL*) in the cavity, i.e. the coupling that gives perfect matching of the cavity circuit in the presence of beam is calculated from the expcted beam-loading (see Eq. 5-1). In the $\beta b<1$ section, the cavity *QL* used in the calculation differs slightly from the design (or optimum), as shown in the figure above and as given in Eq. (5-1). In driving the cavity slightly off the optimum coupling condition (and modulating the supplied power by a similar factor), the differences in beam-loading between the cavities in a module can be partially compensated for. The off-optimum external Q also slows down the cavity filling. Therefore the filling starts earlier in the $\beta b<1$ sectors. The corresponding cavity forward power modulation is obtained through the *Attenuation* (*a*) factor.

$$QL_j^{opt} \approx \frac{Vcav_j}{2\,Ib\,r_j}, \qquad QL_j = \frac{QL_j^{opt}}{\left(T'sync_j\right)^{0.2}}, \qquad a_j = \frac{1}{\sqrt{T'sync_j}} \tag{5-1}$$

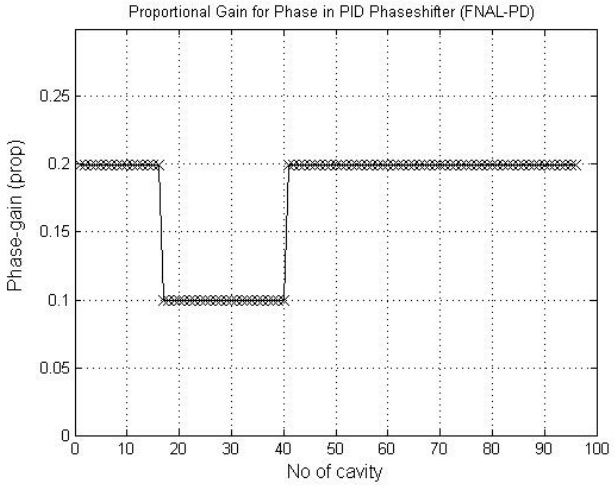
For the same reason the nominal set field in the 16 cavities of the first module was modulated with $1/(T'sync)^{0.5\text{-}0.8}$.

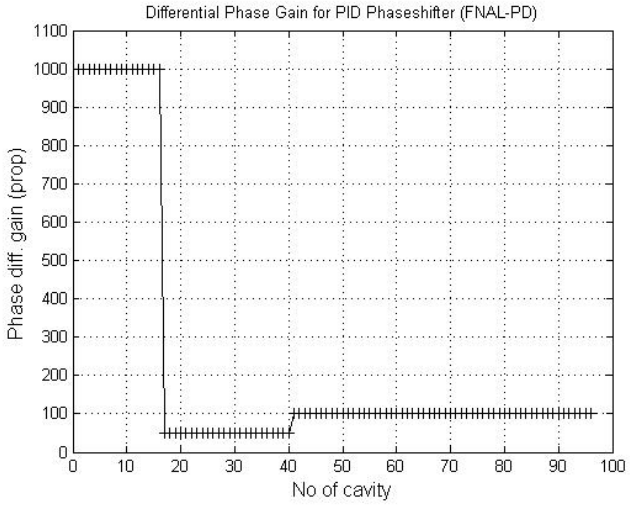
Some of the linac parameters shown in the above plots are the result of extensive design efforts, such as the number of cavities per module, the number of cavities of a particular βc, the number of cells in each particular cavity and their shunt impedance. We therefore cannot discuss these choices in detail here.

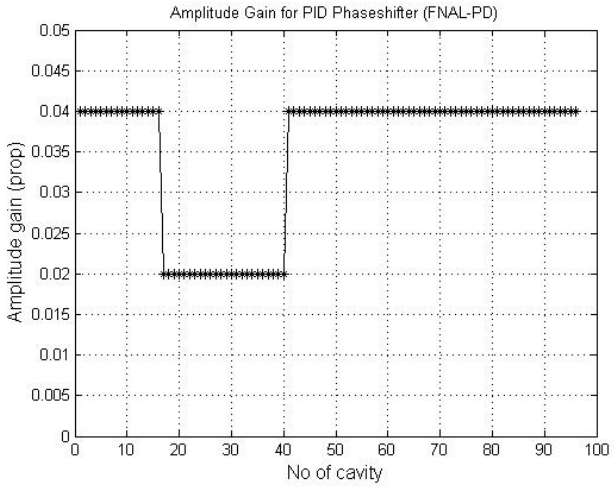
The Lorentz-force detuning and microphonics characteristics in the βc=1 section are those measured in the TTF linac. For the βc<1 cavities the characteristics measured in the SNS cavities were used as a basis. As a general trend the Lorentz-force detuning constants increase in lower frequency cavity designs. Since the accelerating fields also decreases in these cases the overall detuning tends to remain more or less the same for the entire linac. The same is true for micro-phonics. The feed-forward pre-detuning *dw* was calculated from the square of the nominal cavitiy field (accelerating voltage) and the input Lorentz-detuning constants *KL0*.

The beam phase-advance settings were chosen based on those proposed for SNS operation.

The second input structure is *Phaseloop*. The *Phaseloop* parameters only apply to the 96 cavities of the β<1 section where fast vector-modulators are currently being proposed. The vector-modulator parmeters used are the proportional and differential phase and ampitude gains. They are shown in the plots below. All other parameters, such as the phaseshifter operating point *ψ0* and the phase-shifter saturation *ψsat*, are hard-coded into **initphaseloop.m** (and they are listed in Table 5-2).







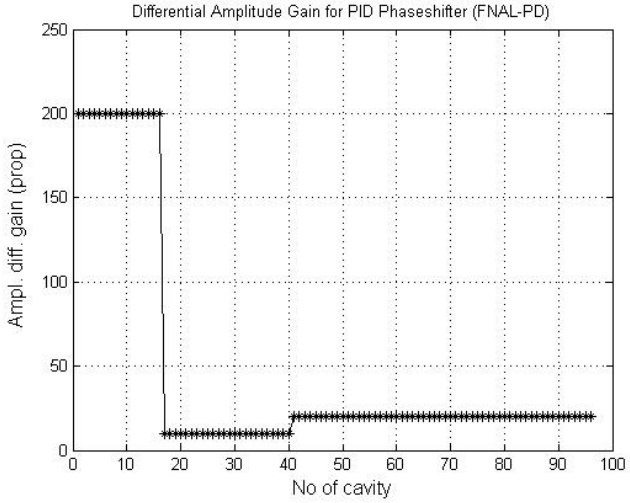


Table 5-2: Constant input parameters for simulation of the Phaseshifter in the S.C.R.E.A.M Proton-Driver simulation.

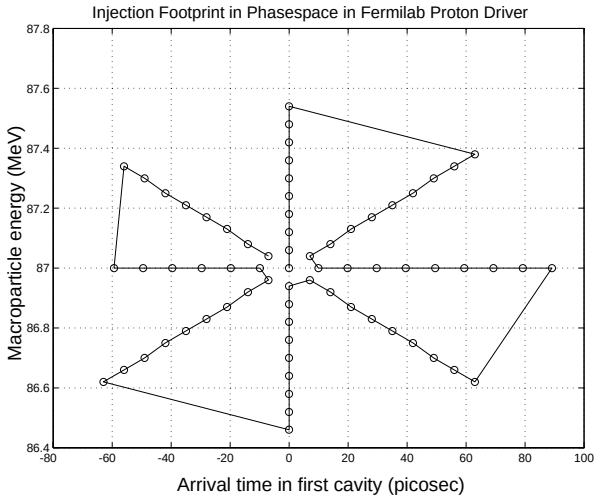
Field	symbol	value
PLSat	ψsat	$\pi/4$
PLSatn	2ψsat/π	1/2
PLIni	ψ0	$\pi/4$
PLsin	sin(ψ0)	sin(PLIni)
PLTau	τps	150 μsec
PLdel	PLdel	O μsec
PLIna	cos(ψ0)	cos(PLIni)

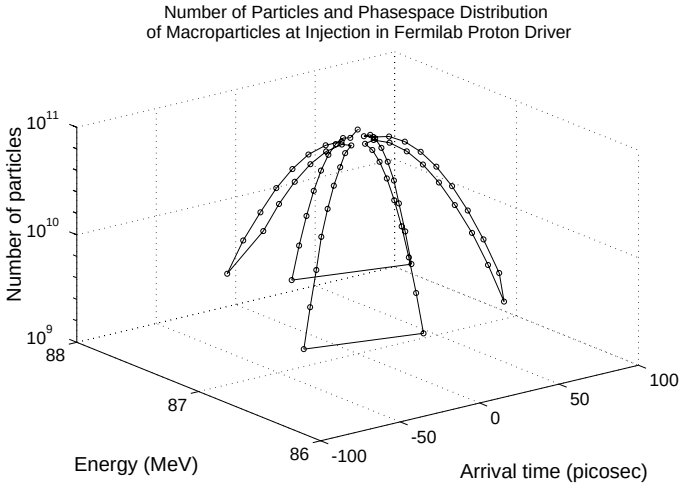
The third set of plots describes the beam parameters. In particular the distribution of the macro-particles in phase-space at injection is shown. Three macro-particles were removed from the distribution (at

(-(7-9) sigma in *tini*) because they tended to get lost during acceleration. The particles simulated are protons (charge 1, mass 938 MeV). The parameters are given in the plot titles. For details consult chapter 3. The 69 macro-particles contain a total of 1.5×10^{11} particles. The total pulse current therefore is 1.5×10^{11} particles per mirco-second, or 25 mA.

Table 5-3: Constant input parameters for the simulation of the beam injection jitter in the S.C.R.E.A.M PD simulation.

Field	value
Input time	O sec
Input energy	87 MeV
Efluc	50 keV
Tfluc	5.8 ps
Ifluc	1%
Ecoherent	50 keV
Tcoherent	5.8 ps
Icoherent	1%
Energy Sigma	60 keV
Time Sigma	9.9 psec
Energy Sigma	60 keV
Sigma Step	1
Nbranch	8
Nσ	9

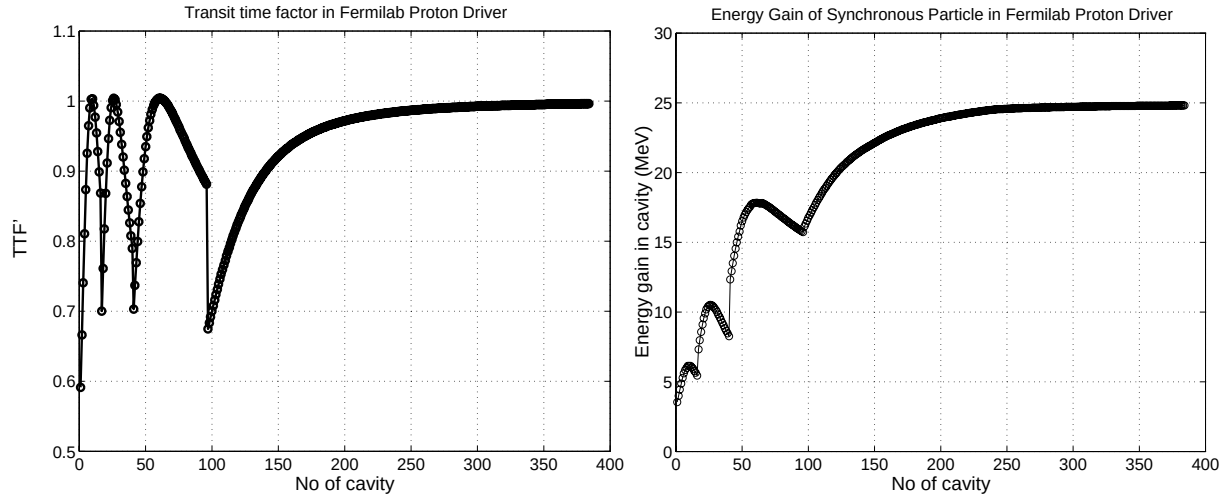


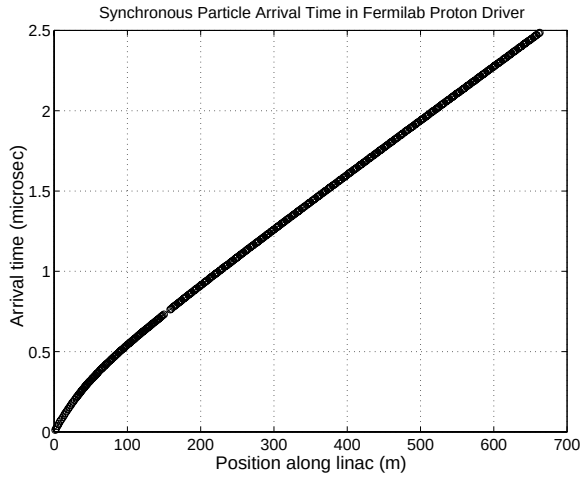


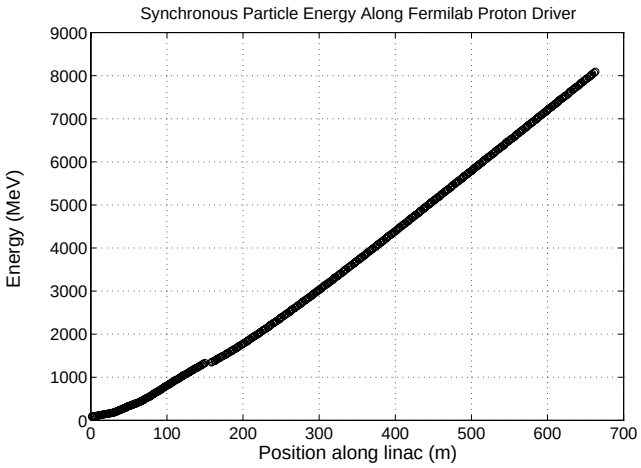
5.3 PreRun Calculation

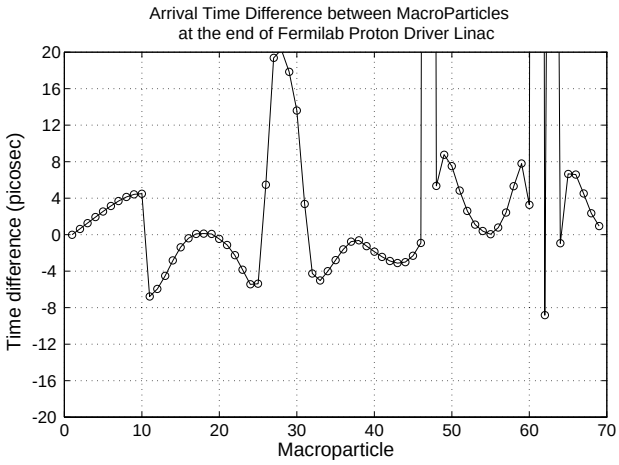
The results of the tracking calculation of the synchronous particle and synchronous bunch are presented in the following plots. These calculations are performed in the **PreRun.m** routine. They assume the nominal accelerating voltage in the cavities, neglecting detuning and beam-loading effects. Of particular interest is the phase-space distribution of the synchronous bunch. The synchronous bunch has the synchronous particle in its center. There are some macro-particles, which appear out of the acceptable range of the energy and arrival time distribution at the end of the linac. These macro-particles are considered to be lost. These are mostly particles, which lag behind the synchronous particle in time and have lower energy. The longitudinal acceptance of the linac appears to be truncated on the side of later injection arrival time, presumably as a result of the negative phase advance of the beam. Efforts were made to reduce the loss fraction and this is the reason why the injection macro-particle distribution is not entirely symmetric (see figures above).

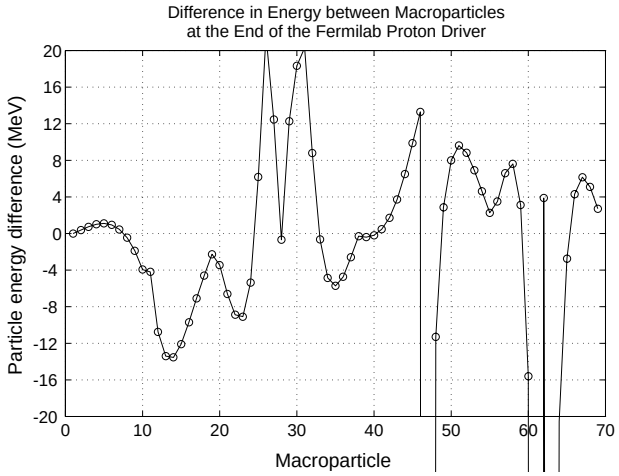
The following shows the transit time factor, energy gain and phase of the synchronous particle as calculated in Prerun. Note that the beam phase advance is included in the calculation. Cavity detuning is not included. Also shown is the phase space distribution for the macro-particles of the synchronous bunch at the start and end of the linac.

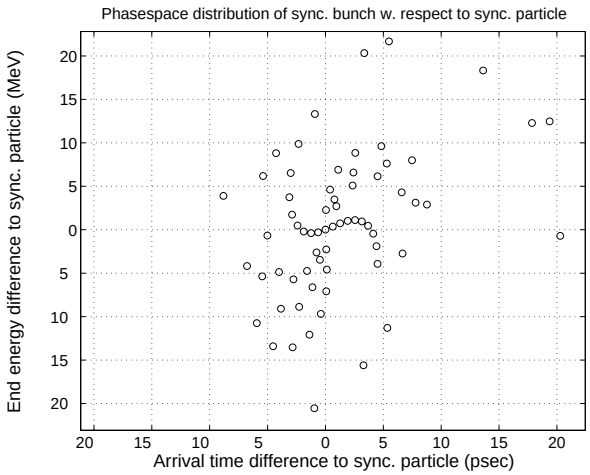


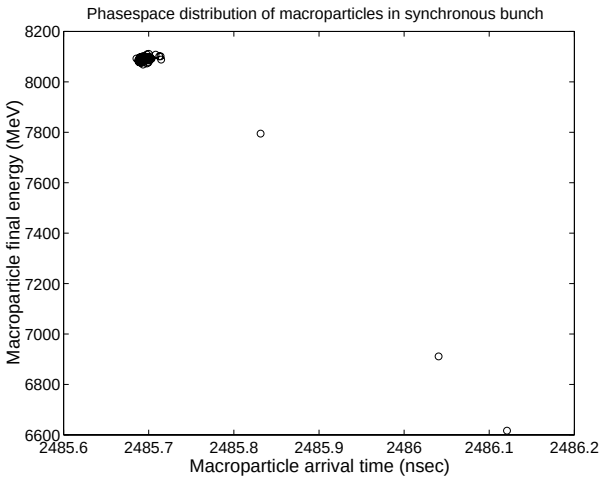

Synchronous Particle Arrival Time in Fermilab Proton Driver
Arrival time (microsec)
Position along linac (m)

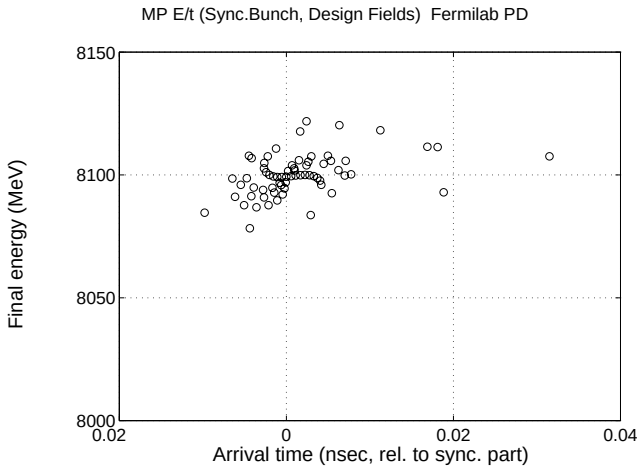

Synchronous Particle Energy Along Fermilab Proton Driver
Energy (MeV)
Position along linac (m)


Arrival Time Difference between MacroParticles
at the end of Fermilab Proton Driver Linac
Time difference (picosec)
Macroparticle


Difference in Energy between Macroparticles
at the End of the Fermilab Proton Driver
Particle energy difference (MeV)
Macroparticle


Phasespace distribution of sync. bunch w. respect to sync. particle
End energy difference to sync. particle (MeV)
Arrival time difference to sync. particle (psec)

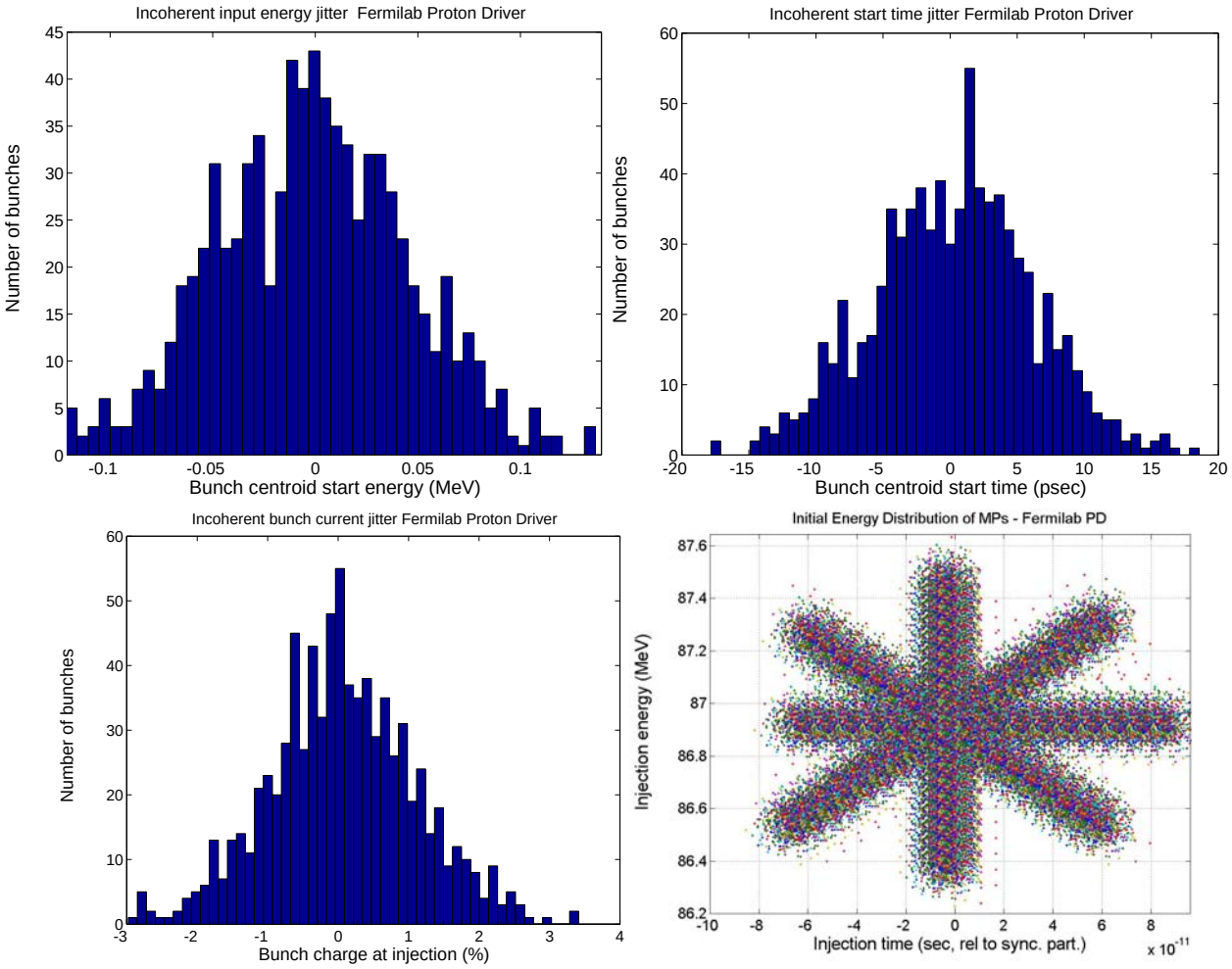

Phasespace distribution of macroparticles in synchronous bunch
Macroparticle final energy (MeV)
Macroparticle arrival time (nsec)



5.4 Full Run - One Pulse

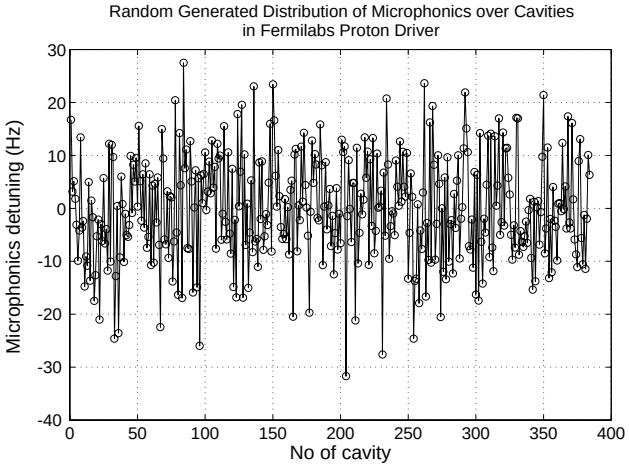
Injection Jitter

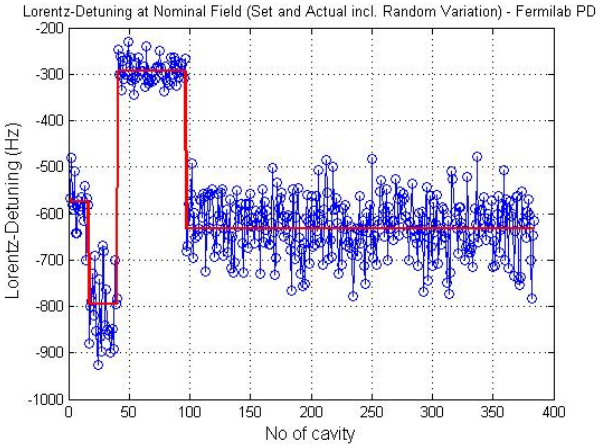
With a beam time of 800 μsec and the 1 μsec step-size the total number of bunches simulated during a pulse is 800. The plots below show a possible random distribution of the bunch centroid start energy and start time for all these bunches. These distributions were calculated by **SimulateField.m**. As discussed in 4.6.), the bunch centroid *E/t/I* at injection is shifted with a coherent (fixed for all bunches) and an incoherent (varying from bunch to bunch) component, both calculated with the MATLAB randn function on the basis of the given distribution widths. The coherent contributions in the case shown were -21.6 keV energy offset (in addition to the 87 MeV start energy), -9.66 psec time-offset (from 0) and a +0.1253% charge increase. The plots below show the incoherent bunch centroid shifts for the 800 bunches in *E/t* and *I*. The last figure shows the E/t injection distribution for all 800x69 macro-particles simulated in one pulse. The distribution also includes the coherent jitter contributions as well as the *E0/t0/I0* values defined in the *Bunches* input. The above is for one pulse and just an example. The calculation results shown below may very well vary for different pulses (the distribution width, however, stays the same).



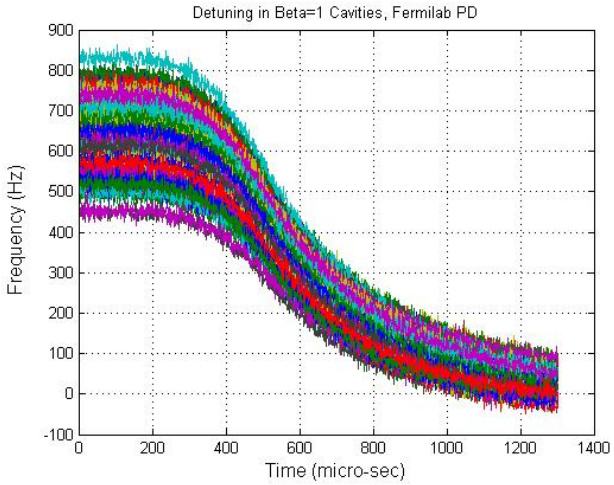
Detuning

Another source of beam jitter is the cavity detuning as a result of Lorentz-forces and micro-phonics. The figure below shows an example of the random variation of the slow micro-phonics detuning in all cavities during one pulse. The fast microphonics variation, which is calculated from a similar distribution, is added to the spectrum below during every time step. The other plot shows the Lorentz-detuning calculated in all cavities at nominal field, clearly showing the fixed and random components. Unlike the micro-phonics distribution, the Lorentz-detuning distribution is invariant throughout the pulse. It is also changing in time, however, as the field amplitude changes in the cavities.





Shown below is the detuning frequency (Hz) in all $\beta c=1$ cavities during a pulse (output parameter *cavpre.dw*). The estimated Lorentz-detuning in the $\beta c=1$ section at 25 MV is -600 Hz. The cavity detuning continues after filling (complete after 500 μsec) because of the mechanical inertia of the cavity (see discussion in section 4/detuning). The Lorentz pre-setting is ~4000/(2π) Hz. In the $\beta c=0.47$ group the estimated Lorentz-detuning at ~7MV=-600Hz. In the $\beta c=0.61$ group the estimated Lorentz-detuning at ~10MV=-600Hz. In the $\beta c=0.81$ group the estimated Lorentz-detuning at ~18 MV = -300 Hz. The fine ripple is (fast and slow) microphonics detuning (10 Hz, HWHM). The figure below shows the detuning profile for all cavities in the $\beta c=1$ sector. The fact that the detuning becomes almost zero toward the end of the pulse indicates that the pre-detuning works successfully. Note the spread in the pre-detuning frequencies at t=0! This spread is introduced in the scream.m script to take into account the random

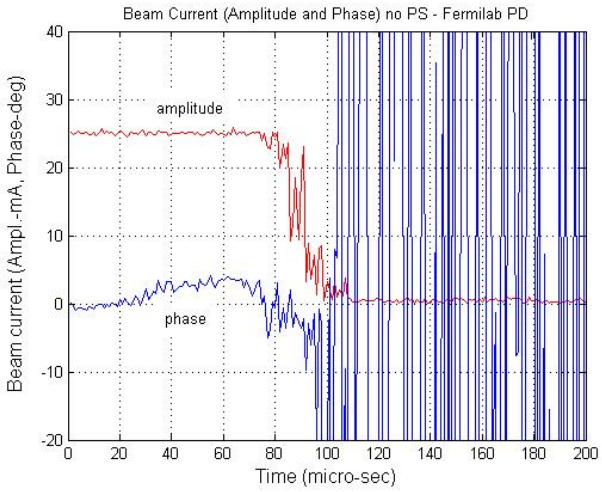


variation of the detuning constants between cavities. This refinement in the pre-detuning is the reason why the spread is reduced at the end of the pulse.

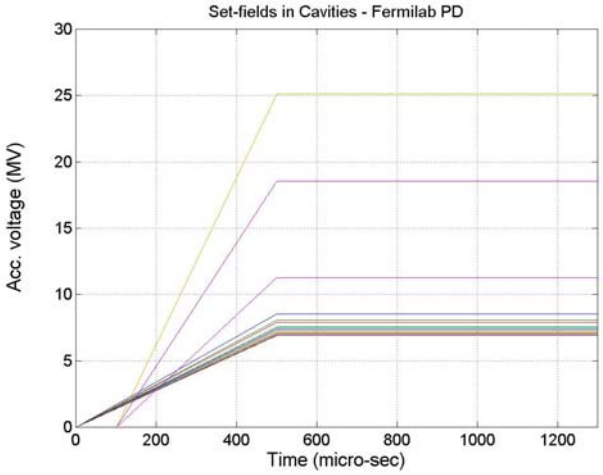
Run without Phaseloop

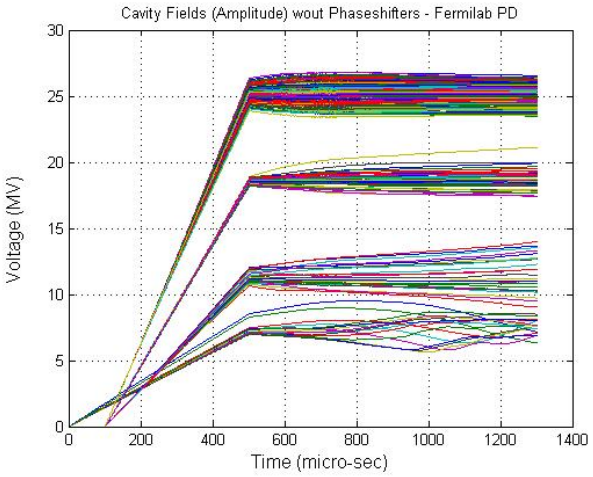
The following shows an example of a pulse simulation without phase-shifters. As is obvious from the graph below, which shows the beam current (*CCur*, amplitude and phase) in the last cavity, the beam does not survive longer than ~50 micro-sec in this condition. The beam phase starts to increase with respect to the synchronous phase (per definition zero), and the current amplitude starts to drop from the nominal 25 mA. The strong phase oscillations after the beam is lost are the result of numerical noise (division through almost zero amplitude).

The *CCur* field constains the sum of the macro-particle currents multiplied with the transit time factors for each macro-particle and a phase-factor containing the phase difference froim each macro-particle to the synchronous particle. The *CCur* array has the dimension *NcavxNb*. The amplitude of CCur therefore essentially is the bunch current for bunches that are running close to the synchronous bunch. When strong deviations from the synchronous phase occur the phase angle increases and the amplitude of CCur drops because of the reduction in transit-time factor. The abovce shown CCur function therefore indicates how well the matching of the bunch and synchronous phase is.

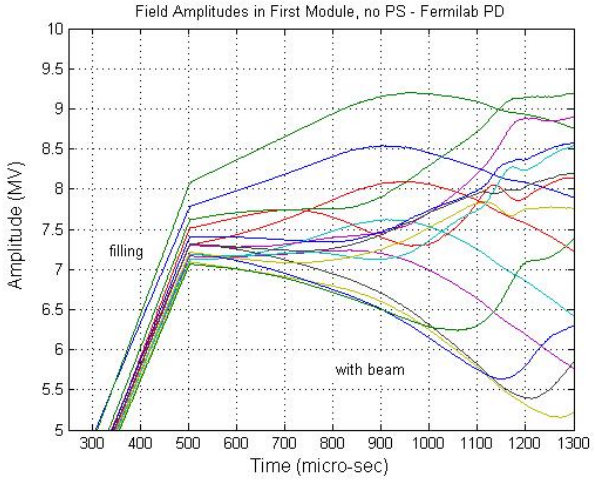


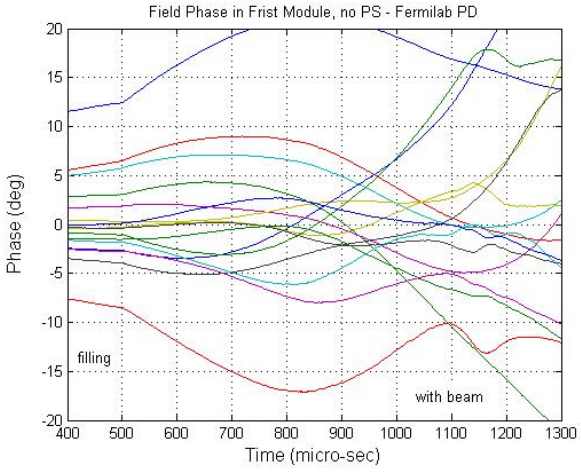
The cause of the current drop is field instability in the low beta modules. Shown below is the field amplitude in all cavities as a function of time. On the left are the set-values, on the right, the actual values. The actual amplitudes depart from the set-values, as a result of beam-loading and detuning as well as the insufficient regulation ability by the vector-sum control system. In the plot below only the $\beta c=1$ section with voltages ~25 MV are well behaved. The fields are unstable in the $\beta c<1$ sections, and most notably in the $\beta c=0.47$ module.





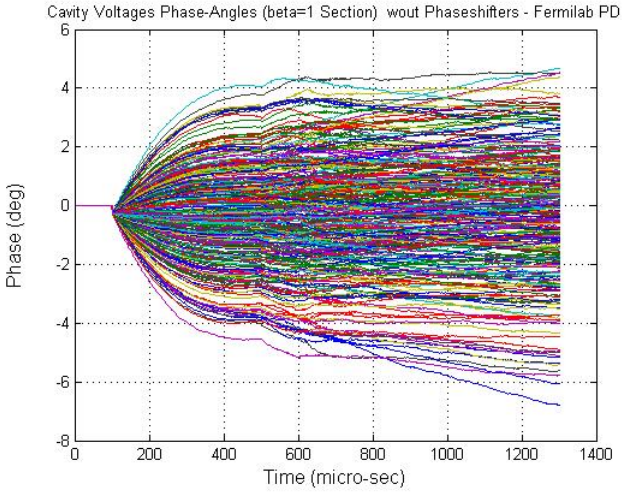
Shown below are the amplitudes and phases (*CField*) in the first 16 cavities (first module) as function of time. The plot clearly shows that the cavity fields start to depart from nominal after ~50 μsecs of beam loading, in some cases even earlier. The vector-sum feedback control

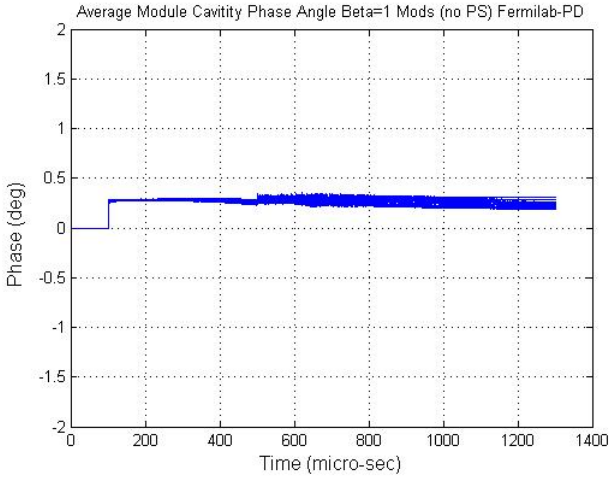




system is not capable of stabilizing the fields sufficiently. Note that reduction in cavity voltage as a result of a phase change α goes with $cos(\alpha)$, thus 20° corresponds to ~10% of loss in amplitude.

The following figures show the phases of the cavity fields. On the left are the actual phases, on the right the phases as averaged over the modules[1]. These plots use the same scale. As expected the average over the module is better behaved by a relative factor *G* (=feedback, ~25). The next set of figures shows the cavities in the $\beta c=1$ sector. The beam is stiff enough in this sector so that variations in beam-loading, detuning, etc.. do not induce significant phase oscillations.

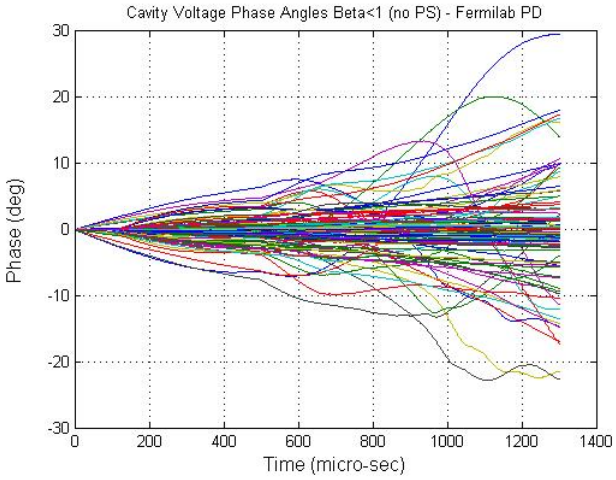


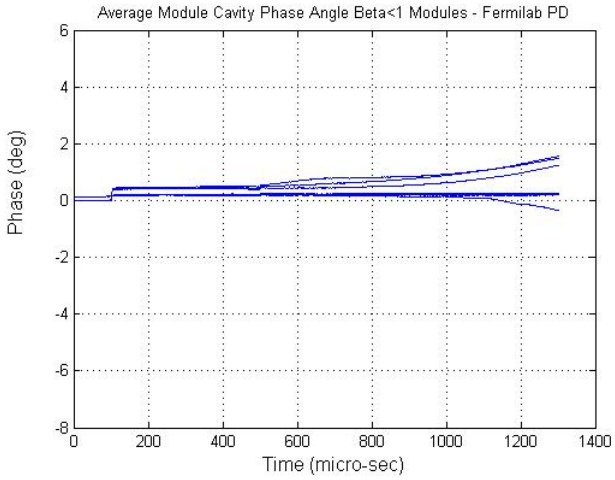


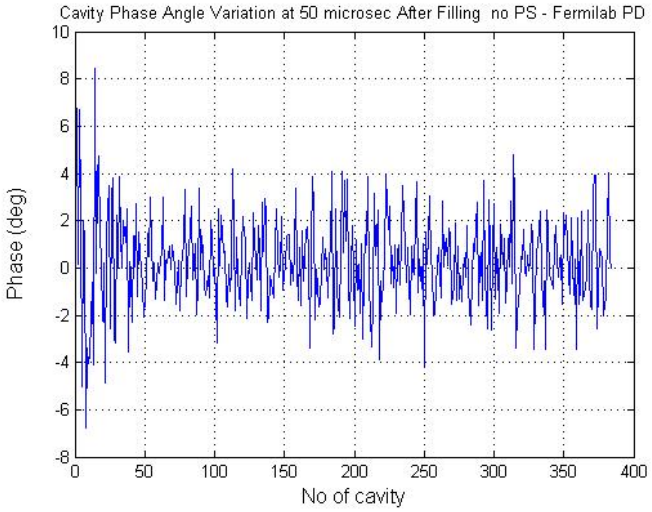
Below are the cavities in $\beta c<1$ sectors, showing again the instability discussed above. The instability even appears at the module level.

The location of the linac where strong cavity phase-fluctuations occur is probably the region where the loss of beam originates. Using the plot below, which shows the phase difference from the synchronous phase (angle(*CField*)) in all cavities at one particular instant in time (50 μsec after filling), one can argue that the instability originates in the first 50 cavities, encompassing the $\beta c=0.47$, $\beta c=0.61$ sections and some cavities from the $\beta c=0.81$ sector.

[1] MATLAB code used to obtain average over module:
```
hold
for n=13:35,
plot((180/pi*sum(angle(cavpre.CField(Mod(n).Cavities,:)))/(Mod(n).N))');
end
```



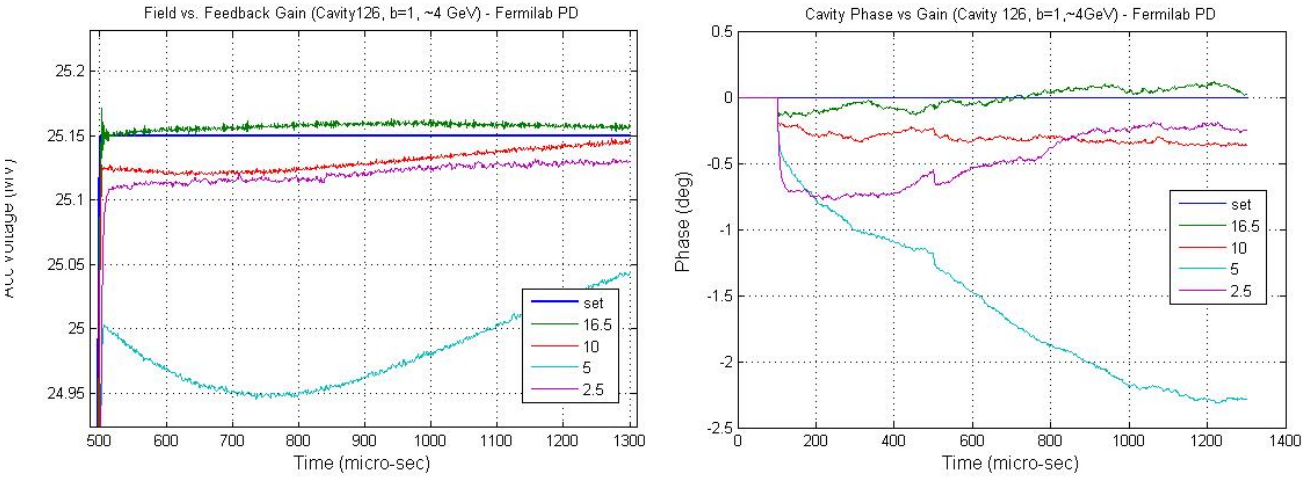




Vector-sum Control

The following discusses the dynamics, and in particular the optimal gain settings of the vector-sum control. Since the vector-sum control is particularly important in the $\beta c=1$ section of the PD, the $\beta c<1$ section of the PD linac was removed (*Eini0*=1336.85 MeV, *tini0*=0sec) and the buinches were all made synchronous (*Efluc*=*Ecoh*=*tfluc*-*tcoh*=*0*) for the purpose of the calculations discussed below. Also the phase-space distribution of the synchronous bunch was strongly compressed ($\sigma t0$=1psec, $\sigma E0$=1keV). No vector-modulators were implemented.

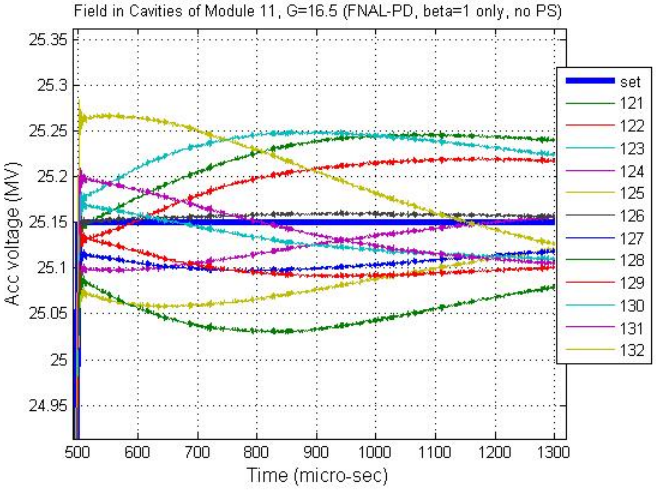
Shown below is the effect of the feedback gain on the field and phase of cavity #126 (in module 11, which contains cavities 121-132). To obtain this plot a special version of scream.m was written, including a loop over the cavity gains[2]. For insufficient gain the fields and phases are very different from their respective set-values. Above the maximum gain shown in these plots the program becomes unstable.

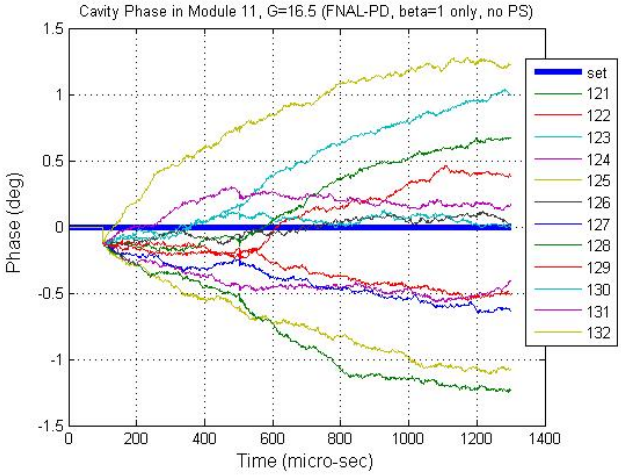


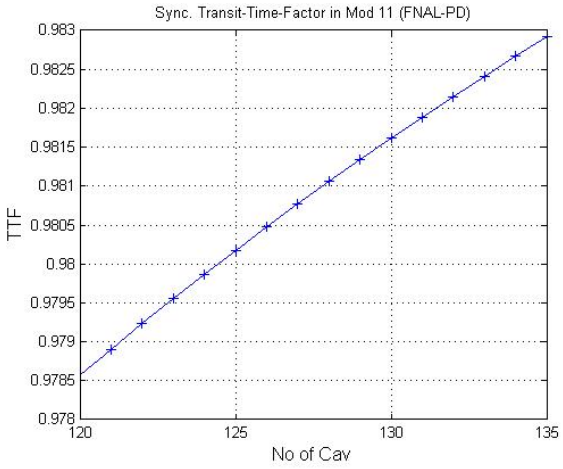
The following figures show the cavity fields and phases in all cavities in the 11th βc=1 module, calculated with the most optimal gain (16.5). We observe that the cavities at the ends of the module have positive amplitude error with beam, cavity in the center come close to the set value, while the other cavities inside the module typically have negative amplitude error.

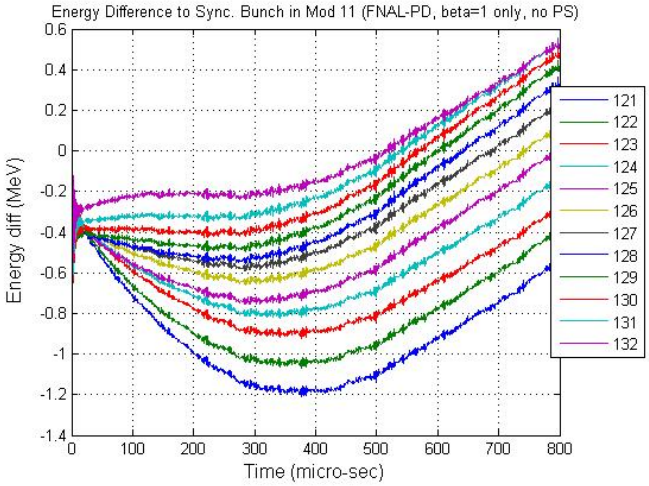
The TTF in this module, shown below also, indicates an increasing beam-loading along the module. The TTF is definitely reflected in the bunch energy-error (below). There are no differences between the beam phases (w. respect to the synchronous phase) in the different cavities. of the module. This beam-phase plot is essentially a short section of the "synchrotron" phase plot discussed in further detail later. Also an energy error plot for the entire linac will be shown later.

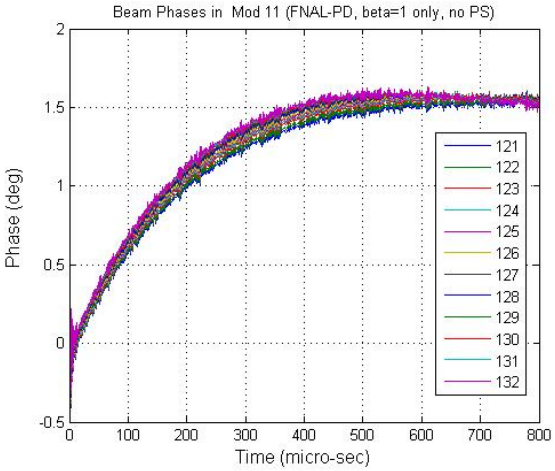
[2] for kf=2:20
SimCav = Cavities;
SimCav.Feedback = SimCav.Feedback/kf;
[cavresult(kf),beamresult(kf)]=SimulateField(SimCav,Mod,Bunches,General,Phaseloop);
end
cr=cavresult;
br=beamresult;


Field in Cavities of Module 11, G=16.5 (FNAL-PD, beta=1 only, no PS)
Acc voltage (MV)
Time (micro-sec)
set
121
122
123
124
125
126
127
128
129
130
131
132

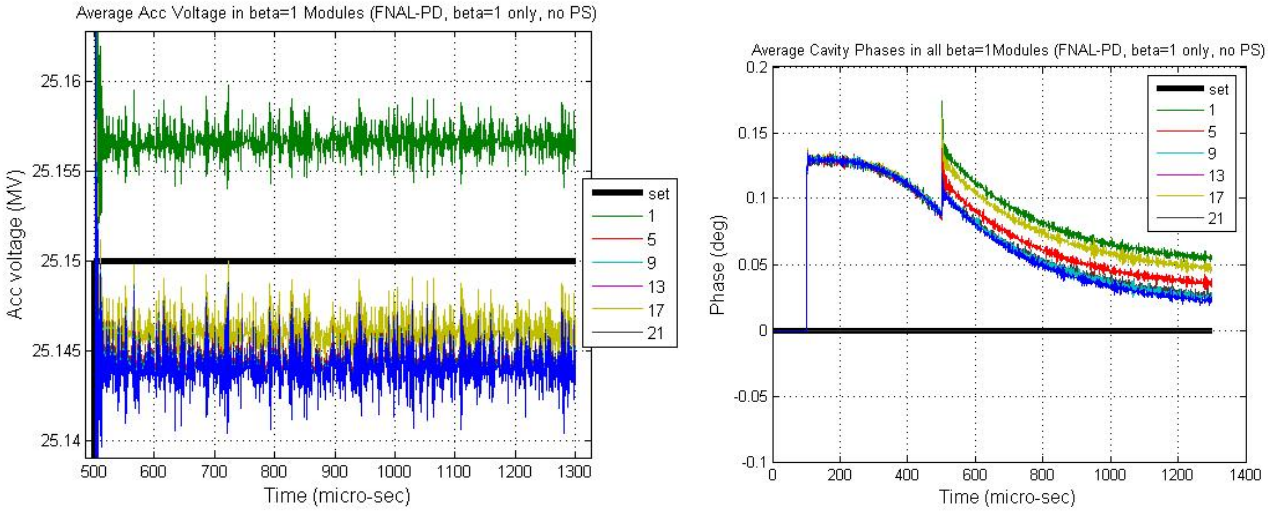

Cavity Phase in Module 11, G=16.5 (FNAL-PD, beta=1 only, no PS)
Phase (deg)
Time (micro-sec)
set
121
122
123
124
125
126
127
128
129
130
131
132


Sync. Transit-Time-Factor in Mod 11 (FNAL-PD)
TTF
No of Cav


Energy Difference to Sync. Bunch in Mod 11 (FNAL-PD, beta=1 only, no PS)
Energy diff (MeV)
Time (micro-sec)
121
122
123
124
125
126
127
128
129
130
131
132

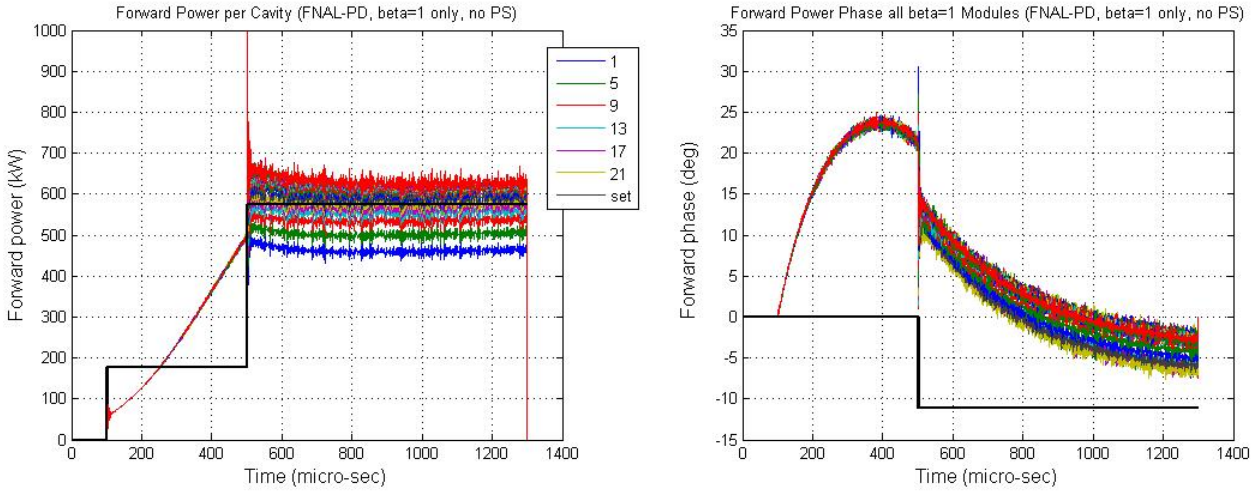

Beam Phases in Mod 11 (FNAL-PD, beta=1 only, no PS)
Phase (deg)
Time (micro-sec)
121
122
123
124
125
126
127
128
129
130
131
132

Since vector-sum control works on the average field in the module it is of interest to verify how successfully it operates on the module level. A special post-processing code was used to compute the average complex field-vectors in the module[3]. As shown in the plot, field (phase) are controlled to within a few % (<1°) at the module level. The amplitude set value is the same in all the modules shown. The perturbation of the cavity phase at the start of beam loading is clearly visible. Toward the end of the pulse the feedback mechanism brings the phase almost back to the synchronous phase.



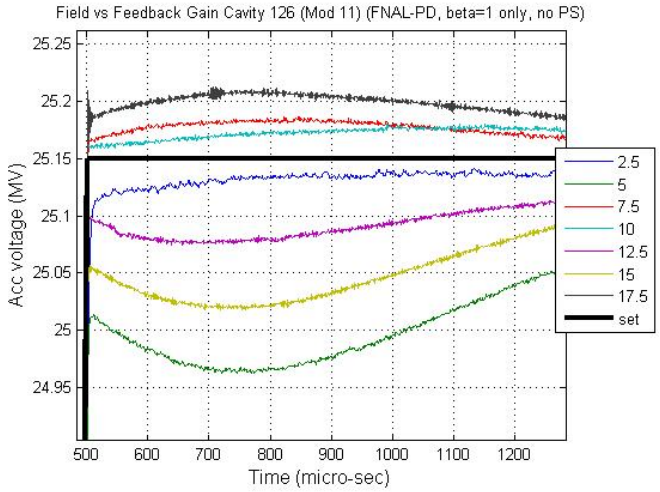
Shown below is the klystron forward power (amplitude and phase, per cavity). It is calculated from 1e+9 CForwd^2. Strictly speaking it is not the forward power as defined as the power entering the cavity coupler, since it does not include the reflected power. During beam loading, however, the reflection is negligible and the shown plots indeed show the forward power from the klystron. Also included is the setvalue (feed-forward component) *SForwd*. The klystron power during beam loading is as expected (power transferred to the beam is ~25MVx25mA=625kW). The phase signal (right plot) clealy shows the Lorentz-force detuning, with the feedback system aiming at matching the klystron phase to the pre-detuned cavity phase (Lorentz-pre-

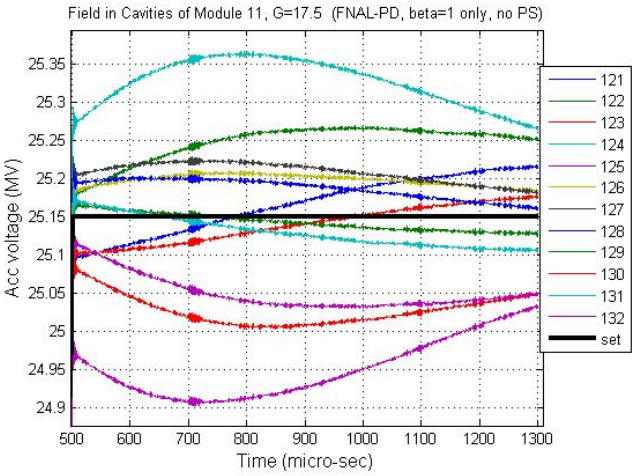
[3] MFM=zeros(6,1300);
 for i=1:4:24,
 MFM(i,:)=zeros(1,1300);
 for k=1:12,
 MFM(i,:)=MFM(i,:)+cr(3).CField((i-1)*12+k,:);
 end
 end
MFM=MFM/12;

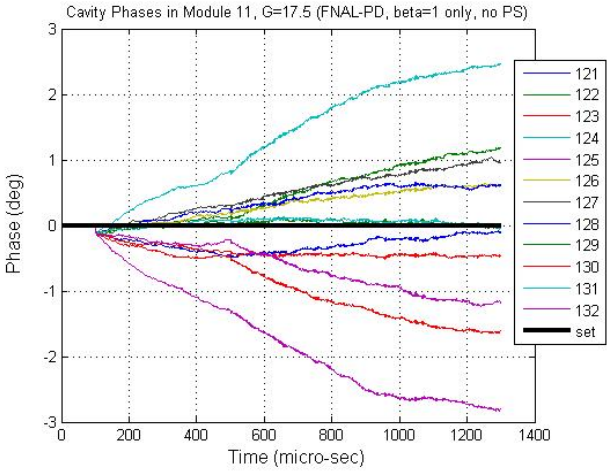


detuning) during filling. Since the Lorentz-force detuning diminishes during filling the feedback finally drives the klystron phase back to smaller values. The expected phase change is $\arctan(\Delta\omega/\omega 12) \sim 10^{\circ}$. The feed-forward phase setting is zero. As the beam arrives the RF phase suddenly jumps by ~-10° to take into account the beam phase-advance (klystron attempts to supply the power at the right phase).

A similar calculation with injection jitter (as defined in the discussion of the input to this simulation, including the random variations) shows similar, albeit slightly worse, results. Shown below are the amplitude and phase in cavity 126 (module 11) for different gain settings. In this case the highest possible gain setting (17.5) is less successful in aligning the actual and set-amplitudes and phases. Also the cavity-to-cavity amplitude and phase-variations within the module, as shown below, are less constrained than in the case with minimal injection jitter.



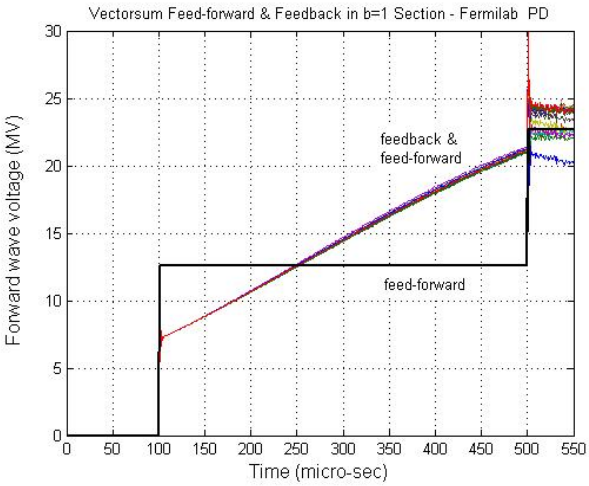


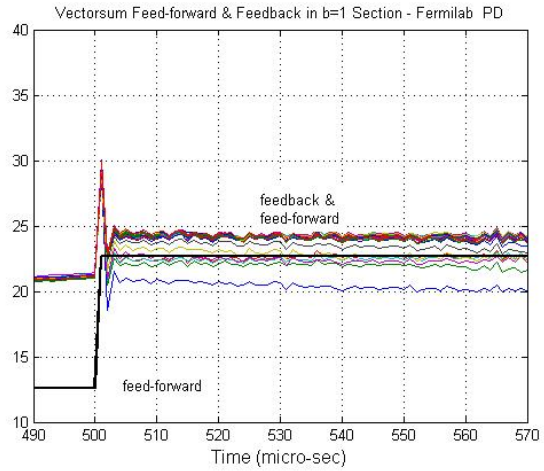


The control signal used by the vector-sum regulation is proportional to the difference between the actual cavity voltage and the set-voltage. Since the vector-sum control averages all signals over the module there is in fact only one signal per module. Furthermore the signal is multiplied with a gain. The S.C.R.E.A.M output parameter *CForwd* contains the amplified feedback signal as well as the feedforward (*SForwd*) signal. Multiplied with the average conversion factor $\overline{\sqrt{2QL_j r_j}}_l$ in module l it becomes an equivalent voltage[4]. Note, however, that the exact amplitude of this voltage is not necessarily relevant since it only serves to regulate the cavity voltage to the nominal value. Since the cavity voltage in the $\beta c=1$ section is 25 MV and the beam current discussed here is 25 mA, the square root of the forward power (~600 kW with beam) is 25 MV. This is a pure coincidence, however! Also shown in the following plots is *SForwd*x$\overline{\sqrt{2QL_j r_j}}_l$, i.e. the feed-forward signal converted to a voltage. Ideally the feed-forward voltage should cover most of the voltage that needs to be supplied to the cavity. Obviously all calculation results

[4] MATLAB code for average conversion factor in module:

```
FieldFac=sqrt(2*Cavities.Qloaded.*Cavities.Rshunt);
for i=1:35,
FieldFacMod(i)=sum(FieldFac(Mod(i).Cavities))/(Mod(i).N);
end
for i=1:1300,
FieldFacModMat(:,i)=FieldFacMod';
end
VForwd=cavpre.CForwd.*FieldFacModMat
```

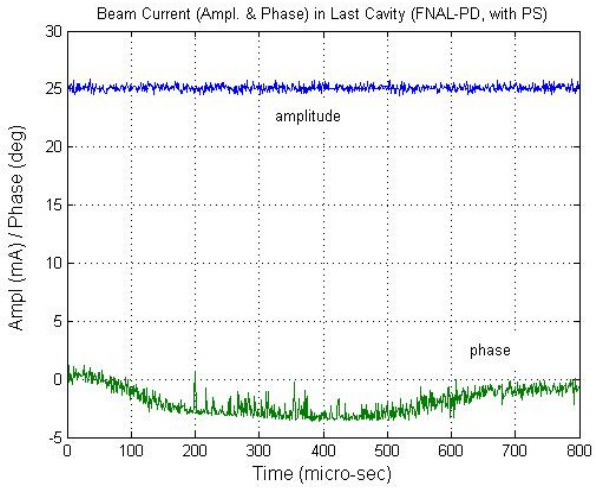


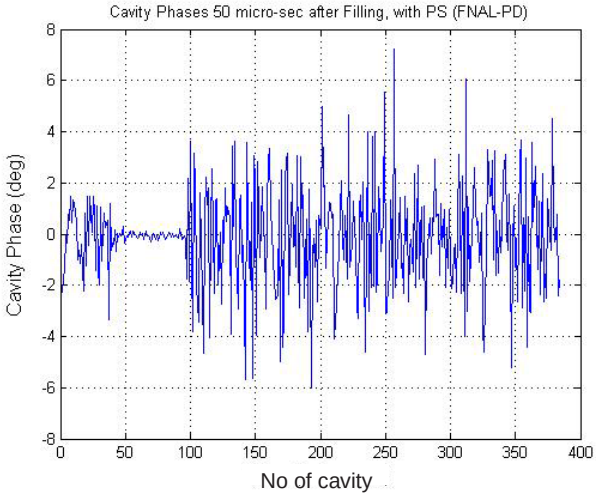


discussed above do not include the effect of the fast ferrite vector-modulator. The following discusses a case with vector-modulators in the βc<1 sectors.

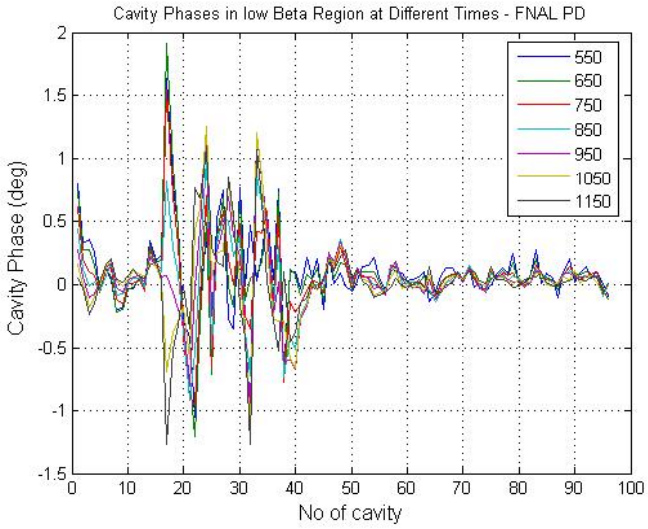
Run with Phaseloop

The following discusses the result of a successful run using vector-modulators in the *βc<1* sectors of the linac. The beam current and phase shown below (in the last cavity) indicate a successful run. The cavity phases at 50 micro-seconds after the end of filling obviously also indicate a much better behaved βc<1 section than before, without the vector-modulators. As a result of successful phase-correction in the βc <1 sectors, the largest phase-variations now occur in the βc =1 sector.

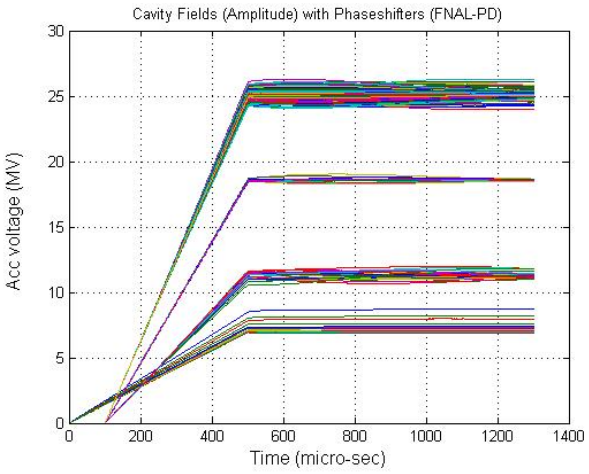


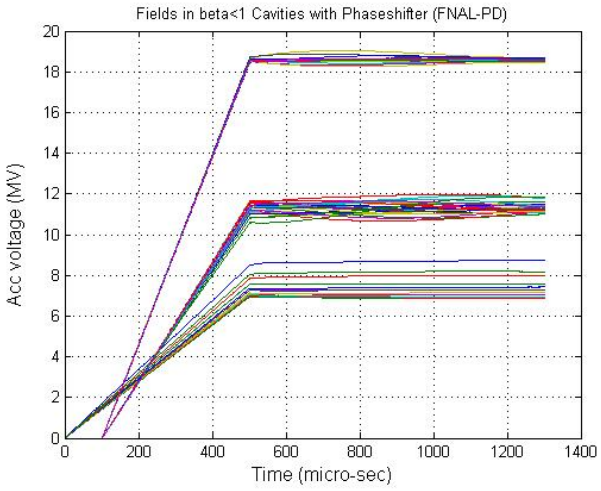


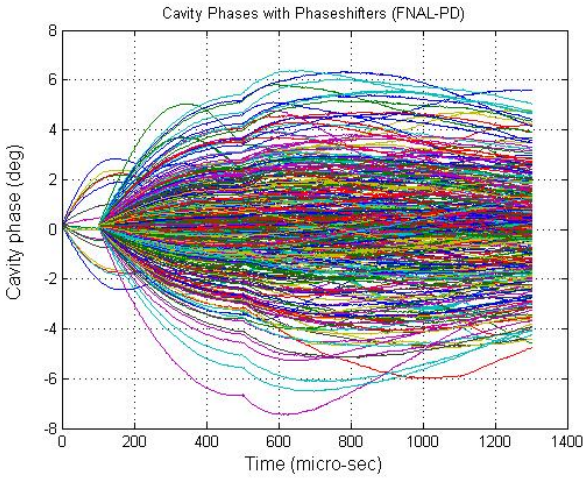
The plot below shows the phases of the βc<1 cavities at different randomly chosen times during the beam pulse, showing that the phases are more or less stable throughout the pulse.

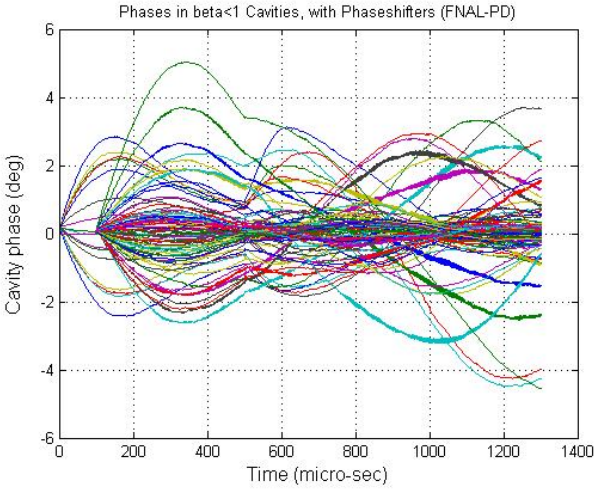


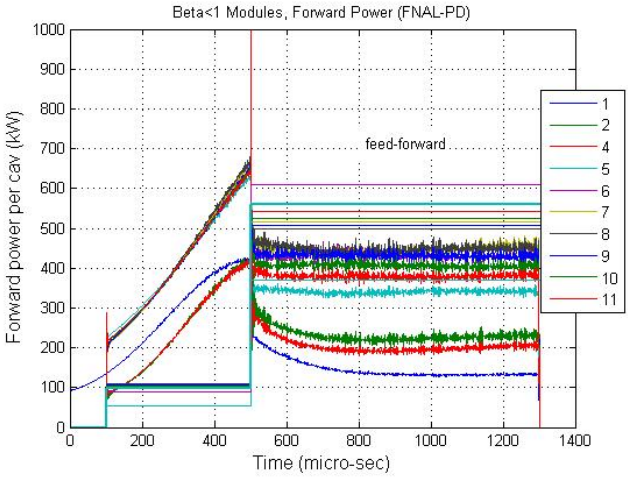
The field amplitudes in the βc<1 cavities are also obviously better behaved than in the case without vector-modulators. Shown are the amplitudes and phases in all cavities (left) and in the βc<1 cavities only (right). The klystron forward power in the βc<1 modules are also shown (amplitude left, phase right) together with the set-values.

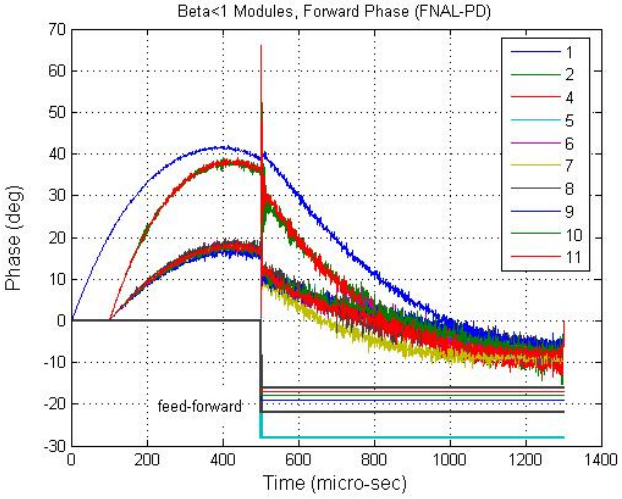






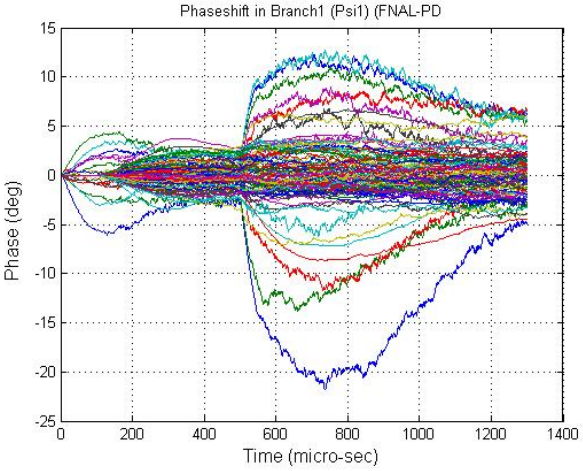


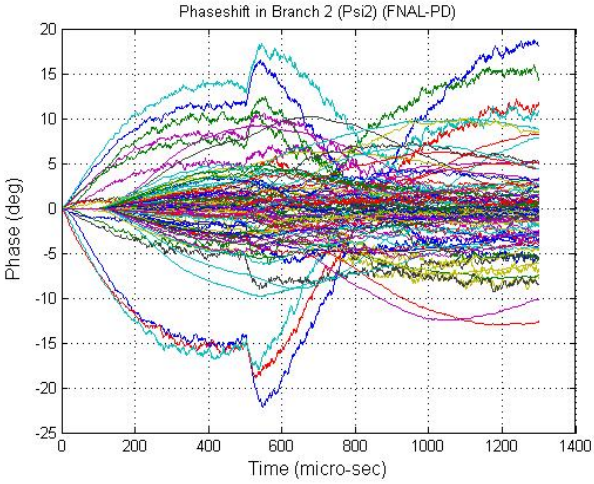




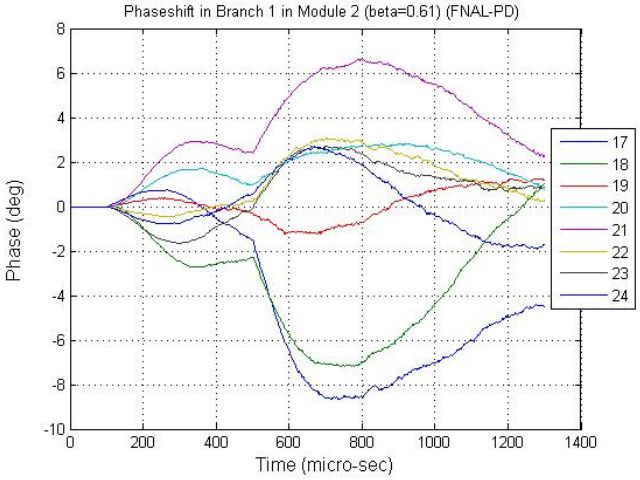
Vector-Modulators

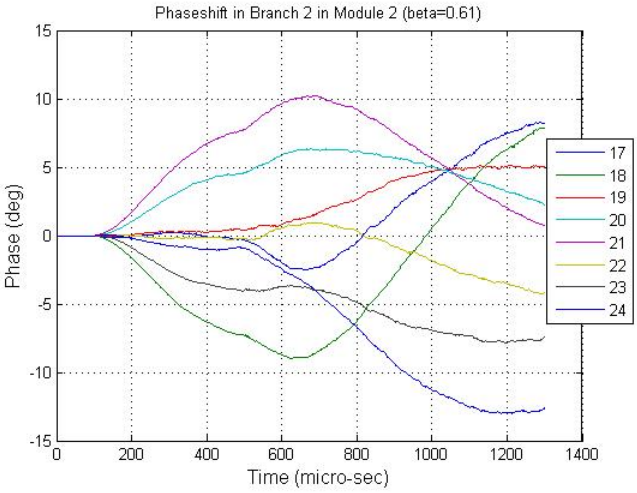
The following shows the phase-shift provided by the two arms of the 96 phase-shifters at all times during the pulse (*cr.sh1* and *cr.sh2*). Except for some outliers the phase signals remain within ±10°. Note, however, that the phase-shifter implementation discussed here operates at the ideal working point ($\pi/4$). In this working point half of the klystron power is reflected from the phase-shifter. This is an un-acceptable situation in a real linac. Better optimized phase-shifter gains and better (faster) control elements are required to allow operation around the 0° working point.

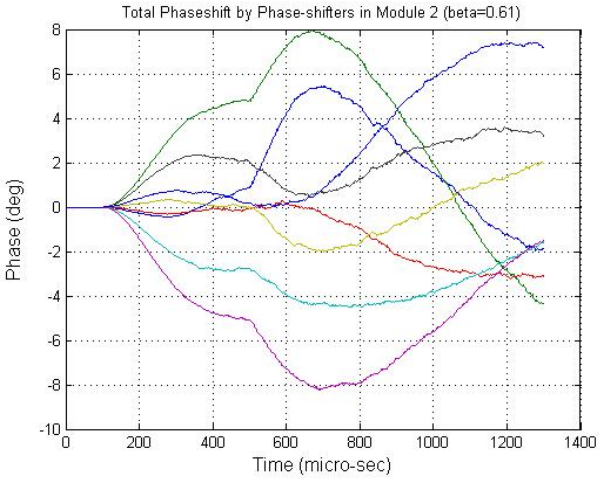




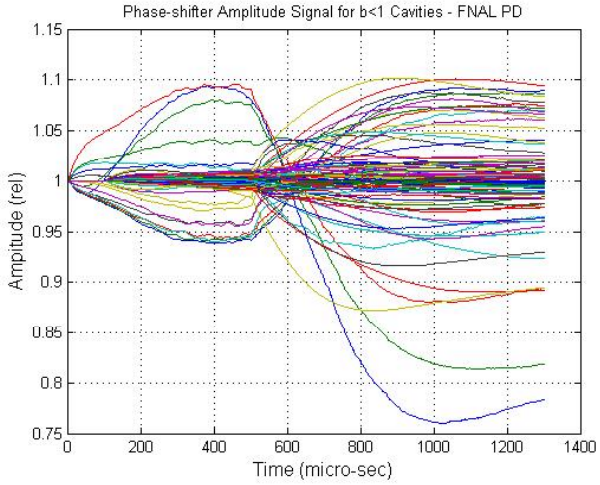
The following shows the phase-shifter angles $\psi 1$ and $\psi 2$ in module 2 ($\beta c=0.61$). Also shown is the total phaseshift (*angle(cr.CFwdpl)*).



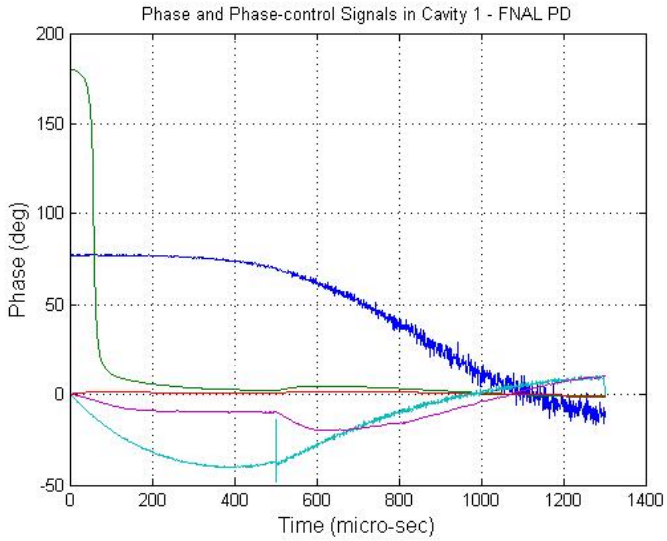




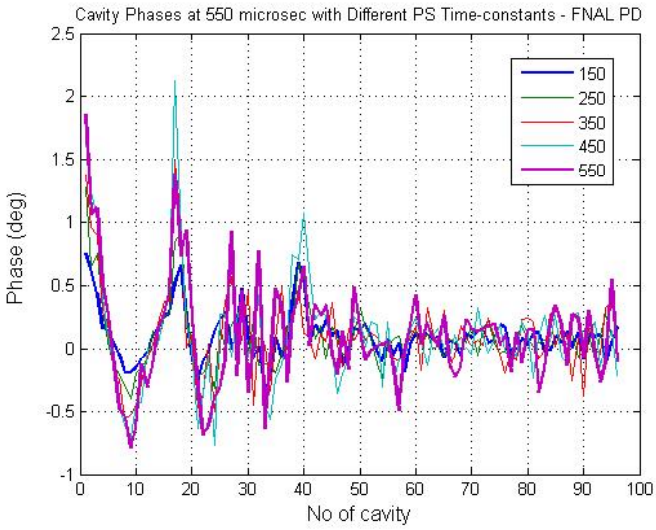
The following plot shows the amplitude of the vector-modulator control signal (*abs(CFwdpl)*). The amplitudes can become larger than one when the working point $\psi 0 \neq 0$.

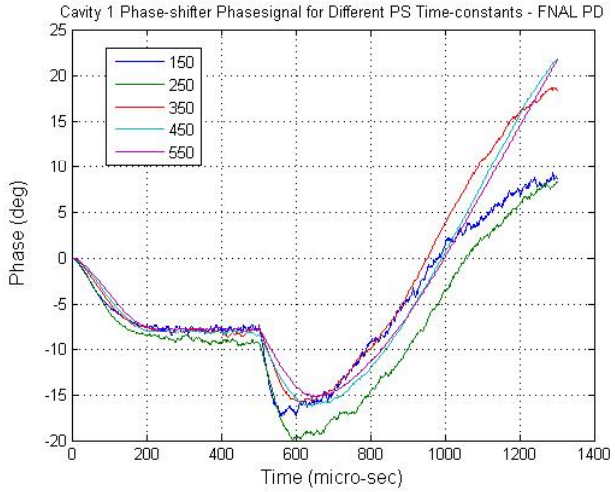


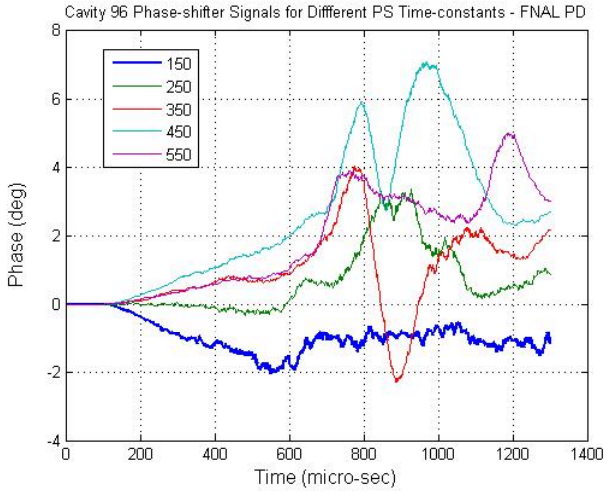
The example below summarizes the effect of the different control signals on the cavity phase (in cavity #1, $\beta c=0.47$). The dark blue curve represents the cavity phase, clearly showing the pre-detuning that is gradually removed as Lorentz-force detuning kicks in. The vector-sum control (light blue) drives the klystron phase to negative values to compensate for the detuning. The fast ferrite vector-modulator (purple) is doing a similar thing. The resulting cavity phase is ~0. The arrival of beam appears in the vector-sum control as a pulse. The fact that the feed-forward signal anticipates beam-loading can be seen in the vector-sum control signal, which is mostly unperturbed by the arrival of the beam.



The following discusses a small study of the effect of the phase-shifter L/R time constant (τps) on the cavity phase-control. Shown below is a series of plots documenting the cavity phases for τps constants between 150 and 550 μsec. The top plot shows the phase in all cavities 50 μsec after beam injection for these different cases. The two plots below show the phase at all times during the pulse in the first and last (# 96) cavity of the $\beta c<1$ sector for the different time constants.



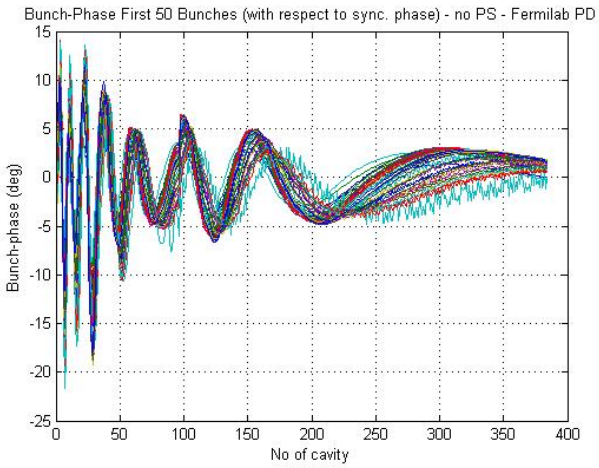


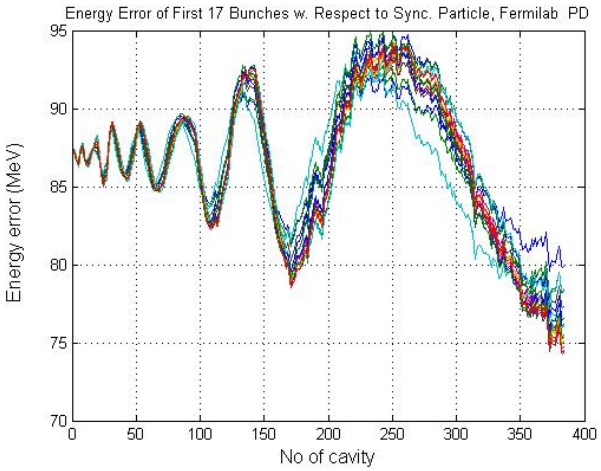


Synchrotron Oscillation

The S.C.R.E.A.M simulation also allows determining the synchrotron oscillation, i.e. the longitudinal oscillation of the bunch with respect to

the synchronous particle. Shown below is the beam-phase (*Angle(CCur)*) for some bunches. This phase is per definition with respect to the synchronous phase. The bunch energy error, *ECur*, (with respect to the synchronous bunch), also reflects the synchrotron motion. The figures below also beautifully demonstrate the Liouville theorem on the conservation of the beam area in phase-space, with the phase-oscillation decreasing as the energy amplitude increases.



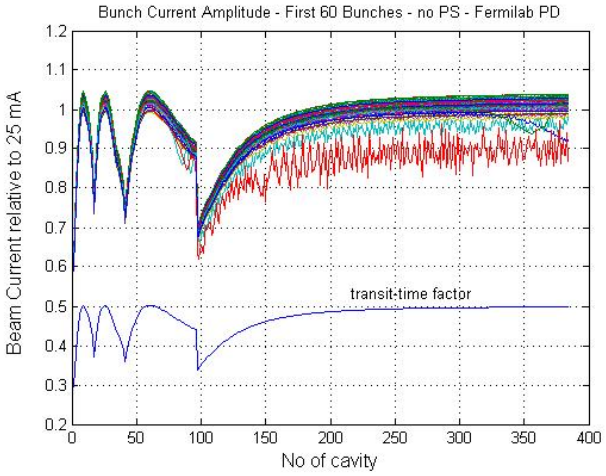


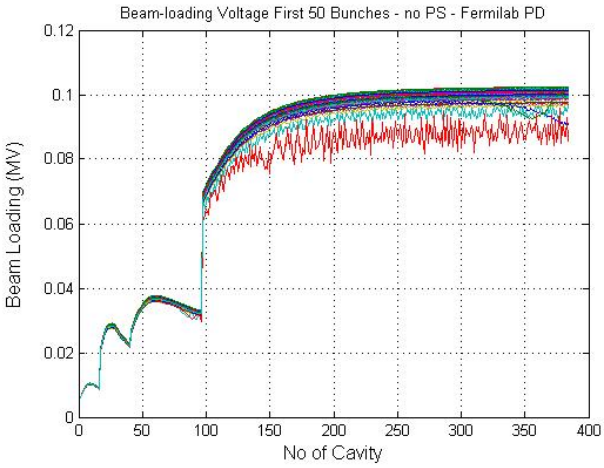
Beam-Loading

The beam-loading is modulated across the module with the transit time factor and vector-sum control can only take care of the mean in the module. Shown below is the amplitude of *CCur*, which clearly reveals the modulation with *T'* (which is also shown in the plot). The beam-loading voltage, shown in the right plot for several bunches is calculated with:

$$\Delta Vb = simbeam = IFac \times CCur = 2\pi \times Frequency \times Rshunt \times \Delta t \times 10^{-6} \times e^{-i\phi 0} \times CCur,$$

where the *IFac* vector had to be augmented to a matrix consisting of 800 identical column-vectors and the so obtained *IFac* matrix had to be multiplied element-wise with the *CCur* matrix.

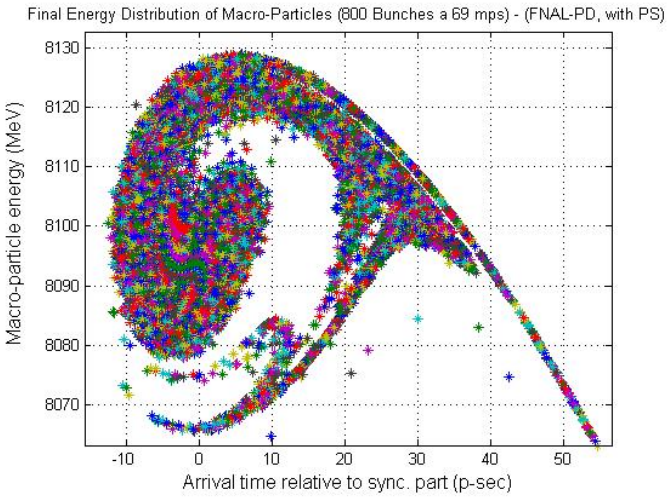


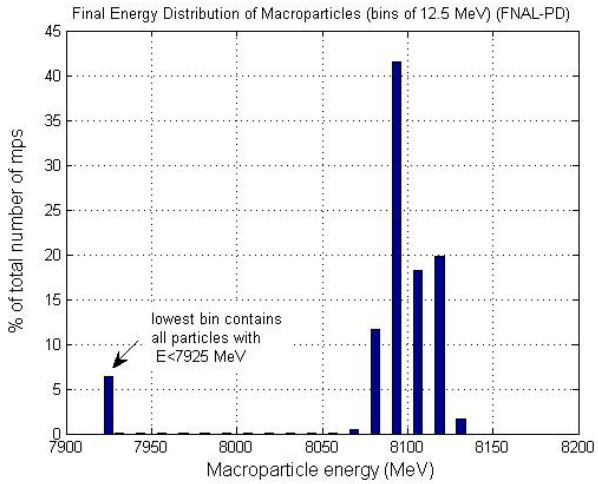


Example Case - Conclusions

The simulations discussed above are intended to document the capabilities of the S.C.R.E.A.M program to simulate RF control strategies in superconducting linacs, such as the Fermilab PD. The PD has several specific characteristics, which were addressed in the version of S.C.R.E.A.M. discussed here. These are related –1- to the question of RF fan-out and the need for fast ferrite vector-modulators and –2- to the intended multi-purpose use of the linac for several particle species (H^- and electrons). Both issues require a careful weigthing of the cost of different technical solutions to the RF controls problem. To reduce the cost of a linac it is, on the one hand, desirable to group as many cavity resonators as possible into one RF circuit, driven by one klystron – modulator unit. The RF phase and amplitude can only be regulated at the output of the klystron and therefore not individually for each cavity. Individual phase and amplitude shifters for each cavity facilitate the optimization of the linac, but they are expensive. Therefore it is desirable to determine the most cost optimal compromise between RF fan-out and the number and complexity of phase shifters. If different beam species are to be accelerated this optimization must include not only intra-pulse control of the RF phase and amplitude but also pulse-to-pulse variations. Simulations with S.C.R.E.A.M as discussed here already indicate that TTF-style vectorsum control is not sufficient to stabilize the cavity fields in the $\beta c<1$ sections of the PD. In the future further fine-tuning of the control algorithms is certainly required to achieve the challenging goals for the final momentum spread (<0.1%), amplitude variation (within $<\pm 0.8$ dB) and phase variation ($<\pm 8$ deg). Note that shifting the cavity phase by the half-bandwidth correspond to a 45 degrees phase-shift!

Shown below is the final beam energy for all 55200 macroparticles accelerated in one pulse (69x800) as calculated in the version that includes the fast vector-modulators. In fact the plot shows plot(*br. Time-(prerun.Time(384,1), br.Energy*), so the time-axis is relative to the time of flight of the synchronous particle. It is clear from the plot that the beam energy in the pulse was ~100 MeV higher than nominal (and that of the synchronous particle). Some macro-particles bunches are lagging behind, producing the tail of the distribution. As shown in the histogram below, however, the tail contains only about 5% of the macro-particles. The beam loss is obviously even smaller since the lost macro-particles are mostly from the fringes of the injection phase-space and therefore hold much less particles. Note that the linac setup was not optimized for minimum beam loss. The histogram was produced with special post-processing code[5].





[5] edgesnew(1)=0;
for u=2:20,
edgesnew(u)=7900+u*12.5;
end
Ehisto=zeros(20,1)
for v=1:800,
Ehisto=Ehisto+histc(bmpre.Energy(:,v),edgesnew);
end
bar(Ehisto)
bar(Ehisto/sum(Ehisto))
bar(100*Ehisto/sum(Ehisto))

6 APPENDIX A

CALCULATION OF THE TRANSIT TIME FACTOR

The transit time factor is used to relate the peak (or average or any other benchmark parameter representing the) accelerating field in the cavity to the effective acceleration of a beam in that cavity. In a single cell cavity the transit time factor describes the effect of the sinusoidal variations of the accelerating field in the cavity as a function of time and space. It also includes the effect of a mismatch between the particle velocity and the cavity design beta, which causes a phase difference between beam and RF fields. In a multi-cell cavity the phase difference as a result of the beta-mismatch increases gradually, from cell to cell. Usually the transit time factor is given for the ideal condition, i.e. for the synchronous particle and therefore does not include the nominal phase difference between beam and RF used for gradient focusing as well as the beam phase errors from synchronous. It also doesn't describe beam-loading effects.

Shown below is the calculation of the transit time factor for a single cell cavity in the case in which the cavity-beta equals the beam-beta ($\beta_b=\beta_c$). The coordinate system for the calculation was chosen such that the center of the cell is at z=0. The spatial field distribution in the cell is described by a cosine function with nodes at the entrance and exit from the cell. The field is highest in z=0 at t=0 when the $cos\omega_{RF}t$ function peaks. As shown below the (normalized) transit time factor for the synchronous particle at speed c in a $\beta=1$ type cavity in the π-mode is 0.5. That means that in the best case the particle sees an accelerating voltage of $0.5E_{max}L_c$ as it crosses the cell of length L_c, where E_{max} is the cavity peak electric field.

$$T=\frac{1}{L_c}\int_{-L_c/2}^{+L_c/2}\left[\cos(\omega_{RF}t)\cos\left(\frac{\pi}{L_c}z\right)\right]dz=\left|\begin{array}{c}t=z/c\\ \omega_{RF}=2\pi f_{RF}=\pi c/L_c\end{array}\rightarrow\omega_{RF}t=\pi z/L_c\right|=\frac{1}{L_c}\int_{-L_c/2}^{+L_c/2}\left[\cos^2\left(\frac{z\pi}{L_c}\right)\right]dz$$

$$\rightarrow\left|x=\frac{\pi}{L_c}z,\,dz=\frac{L_c}{\pi}dx\right|\rightarrow=\frac{1}{\pi}\int_{-\pi/2}^{+\pi/2}\left[\cos^2(x)\right]dx=\left|\int_{-\pi/2}^{+\pi/2}\left[\cos^2(x)\right]dx=\frac{\pi}{2}\right|=\frac{1}{2} \qquad \text{(A-1)}$$

A similar derivation can be obtained for the more general case when the beam β_b and cavity β_c are different. Again the calculation is performed in only one cell:

$$T=\frac{1}{L_c}\int_{-L_c/2}^{+L_c/2}\left[\cos(\omega_{RF}t)\cos\left(\frac{\pi}{L_c}z\right)\right]dz=\left|\begin{array}{c}\varpi_{RF}t=\varpi_{RF}\,z/\beta_b c=\dfrac{\pi\beta_c z}{L_c\beta_b}\\ L_c=\dfrac{\beta_c T_{RF}}{2}=\dfrac{\pi\beta_c c}{\varpi_{RF}}\end{array}\right|=$$

$$=\frac{1}{L_c}\int_{-L_c/2}^{+L_c/2}\left[\cos\left(\frac{\pi\beta_c}{L_c\beta_b}z\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz=\left|\alpha\equiv\frac{\beta_c}{\beta_b}\right|=\frac{1}{L_c}\int_{-L_c/2}^{+L_c/2}\left[\cos\left(\frac{\alpha\pi}{L_c}z\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz$$

$$\frac{1}{L_c}\int_{-L_c/2}^{+L_c/2}\left[\cos\left(\frac{\alpha\pi}{L_c}z\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz=\left|\frac{\pi}{L_c}z=x,\ dz=\frac{L_c}{\pi}dx\right|=\frac{1}{\pi}\int_{-\pi/2}^{+\pi/2}[\cos(\alpha x)\cos(x)]dx=$$

$$\frac{1}{\pi(1+\alpha)}\left[\cos(\alpha x)\sin(x)\Big|_{-\pi/2}^{\pi/2}+\alpha\int_{-\pi/2}^{+\pi/2}[\cos[(\alpha-1)x]]dx\right]=\left|x=\frac{\pi}{L_c}z,\ dx=\frac{\pi}{L_c}dz\right|=$$

$$=\frac{1}{\pi(1+\alpha)}\left[\cos\left(\frac{\alpha\pi}{L_c}z\right)\sin\left(\frac{\pi}{L_c}z\right)\Big|_{-L_c/2}^{L_c/2}+\frac{\alpha\pi}{L_c}\int_{-L_c/2}^{+L_c/2}\cos\left((\alpha-1)\frac{\pi}{L_c}z\right)dz\right]=$$

$$=\frac{1}{\pi(1+\alpha)}\left[\cos\left(\frac{\alpha\pi}{L_c}z\right)\sin\left(\frac{\pi}{L_c}z\right)\Big|_{-L_c/2}^{L_c/2}+\frac{\alpha}{\alpha-1}\sin\left[(\alpha-1)\frac{\pi}{L_c}z\right]\Big|_{-L_c/2}^{L_c/2}\right]=$$

$$=\frac{1}{\pi(1+\alpha)}\left[2\cos\left(\frac{\alpha\pi}{2}\right)+\frac{2\alpha}{\alpha-1}\sin\left[(\alpha-1)\frac{\pi}{2}\right]\right]=\frac{2\cos\left(\alpha\,\pi/2\right)}{\pi(1+\alpha)(1-\alpha)}=$$

$$=\frac{2\cos\left(\alpha\,\pi/2\right)}{\pi(1-\alpha^2)}=\left|\alpha=\frac{\beta_c}{\beta_b}\right|\ \Rightarrow\ T=\frac{2\cos\left(\frac{\pi}{2}\frac{\beta_c}{\beta_b}\right)}{\pi\left(1-\left(\frac{\beta_c}{\beta_b}\right)^2\right)}$$

(A-2)

The transit time function as calculated above, unfortunately cannot be implemented in a computer program because it generates a singularity for $\beta_c=\beta_b$ instead of producing *0.5*. A calculation using de L'Hopital's rule would produce the expected value. In the S.C.R.E.A.M code, however, the cos-term was Taylor expanded to prevent this problem. First the cos term was transformed into a sin term to allow for a Taylor expansion around zero. This trick automatically yields

series terms canceling against the denominator, thus removing the singularity.

$$T = \frac{2\cos\left(\frac{\pi}{2}\frac{\beta_c}{\beta_b}\right)}{\pi\left(1-\left(\frac{\beta_c}{\beta_b}\right)^2\right)} = \frac{2\sin\left[\frac{\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}{\pi\left(1-\left(\frac{\beta_c}{\beta_b}\right)^2\right)} = \left|x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + \ldots\right| =$$

$$= \frac{2}{\pi\left(1+\left(\frac{\beta_c}{\beta_b}\right)\right)\left(1-\left(\frac{\beta_c}{\beta_b}\right)\right)}\left\{\frac{\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right) - \frac{\pi^3}{3!2^3}\left(1-\frac{\beta_c}{\beta_b}\right)^3 + \frac{\pi^5}{5!2^5}\left(1-\frac{\beta_c}{\beta_b}\right)^5 + \ldots\right\}$$

$$T \approx \frac{1}{\left(1+\frac{\beta_c}{\beta_b}\right)}\left[1-\left(\frac{\pi^2}{24}\right)\left(1-\frac{\beta_c}{\beta_b}\right)^2 + \left(\frac{\pi^4}{1920}\right)\left(1-\frac{\beta_c}{\beta_b}\right)^4 + \ldots\right]$$

(A-3)

Figure 6-1 shows a comparison of *T* calculated with the approximation (A-3) and with the exact calculation using (A-2).

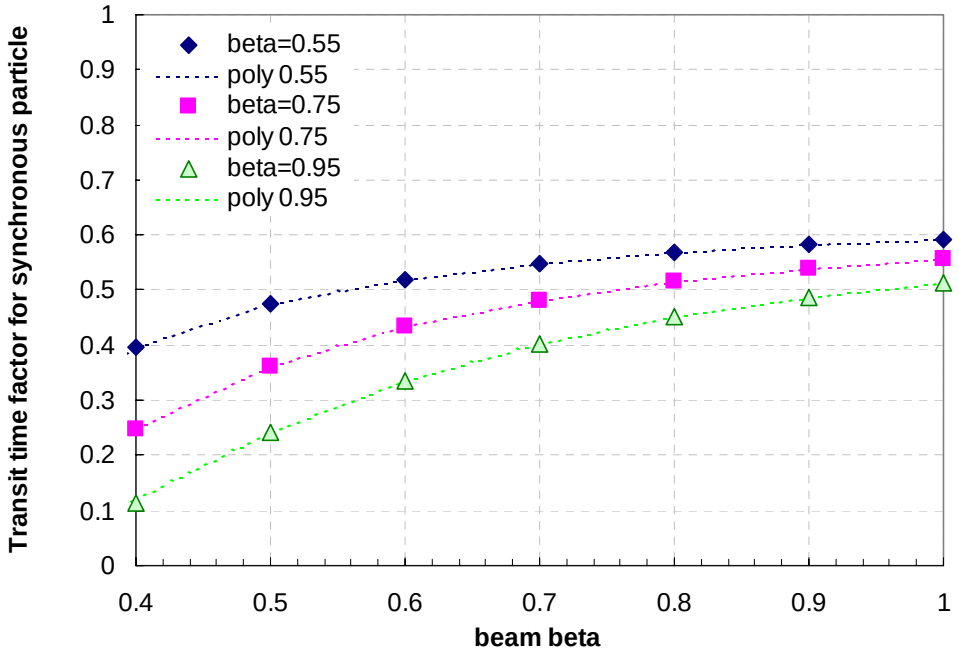


Figure 6-1: Ideal transit time factor for the synchronous particle of varying beta in single cell cavities with different betas: complete model (Eq. *A-2*) and polynomial approximation (Eq. *A*-3).

To use (A-3) in the calculation of *T* in the multi-cell cavity, it is necessary to prove that the phase difference, which appears after one cell (and accumulates from cell to cell) due to the β–mismatch, can be extracted from the transit-time-factor integral. Also, the procedure used in the program requires that the non-synchronous particle can be simulated on the basis of the transit time factor of the synchronous particle. That requires that the phase factor be extracted from the transit time factor integral at any stage of the calculation. The following repeats the above calculation, except that an additional phase factor $\Delta\phi$ is introduced into the argument of the $cos\omega t$ function.

$$T = \frac{1}{L_c}\int_{-L_c/2}^{+L_c/2}\left[\cos\left(\omega_{RF}t + \Delta\phi\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz = \left|\begin{array}{l}\cos\left(\omega_{RF}t + \Delta\phi\right) = \\ \cos\left(\omega_{RF}t\right)\cos\left(\Delta\phi\right) - \sin\left(\omega_{RF}t\right)\sin\left(\Delta\phi\right)\end{array}\right| =$$

$$= \frac{1}{L_c}\cos\left(\Delta\phi\right)\int_{-L_c/2}^{+L_c/2}\left[\cos\left(\omega_{RF}t\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz - \frac{1}{L_c}\sin\left(\Delta\phi\right)\int_{-L_c/2}^{+L_c/2}\left[\sin\left(\omega_{RF}t\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz$$

(A-4)

The first integral above corresponds to the phase-free transit time factor as calculated above in (A-2). As will be shown below the second integral on the right is zero.

$$\int_{-L_c/2}^{+L_c/2}\left[\sin\left(\alpha\frac{\pi}{L_c}z\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz = \left|partial\ \mathrm{int}\right| =$$

$$= \frac{L_c}{\pi}\sin\left(\frac{\pi}{L_c}z\right)\sin\left(\alpha\frac{\pi}{L_c}z\right)\Bigg|_{-L_c/2}^{+L_c/2} - \alpha\int_{-L_c/2}^{+L_c/2}\sin\left(\frac{\pi}{L_c}z\right)\cos\left(\alpha\frac{\pi}{L_c}z\right)dz$$

(A-4')

The first term above integrates to zero. The integral can be transformed with:

$$\sin\left(\frac{\pi}{L_c}z\right)\cos\left(\alpha\frac{\pi}{L_c}z\right) = \sin\left[\frac{\pi}{L_c}\left(1+\alpha\right)z\right] - \cos\left(\frac{\pi}{L_c}z\right)\sin\left(\alpha\frac{\pi}{L_c}z\right),$$

(A-4'')

which includes the original integral. Thus the following transformation can be made:

$$\int_{-L_c/2}^{+L_c/2}\left[\sin\left(\alpha\frac{\pi}{L_c}z\right)\cos\left(\frac{\pi}{L_c}z\right)\right]dz=\frac{\alpha}{\alpha-1}\int_{-L_c/2}^{+L_c/2}\sin\left[\frac{\pi}{L_c}(1+\alpha)z\right]dz=0$$

(A-4''')

The sin function integrates out to zero. Thus it can be concluded that:

$$T(\beta_c,\beta_b,\Delta\phi)=\cos(\Delta\phi)T(\beta_c,\beta_b) \qquad \text{(A-5)}$$

To generalize the transit time factor to N_c cells the phase difference accumulated by a particle in the course of its travel from the middle of one cell to the middle of the next cell needs to be taken into account.

$$\Delta\phi=\pi-\omega_{RF}\frac{L_c}{\beta_b c}=\pi-\pi\frac{\beta_c}{\beta_b}=\left(1-\frac{\beta_c}{\beta_b}\right)\pi\quad(rad) \qquad \text{(A-6)}$$

The above phase difference also takes into account that neighboring cells are generally operated with a phase difference of π (in the π-mode). Furthermore, a convention needs to be made regarding the reference position, i.e. the location in the multi-cell cavity where the phase is arbitrarily set to zero. In S.C.R.E.A.M the reference point lies exactly in the middle of the cavity, coinciding with the middle of the middle cell for the case of an odd number of cells and in the middle between two center cells when the number of cells is even.

Taking into account (A-5), the multi-cell transit time factor can be written as:

$$T=\frac{1}{N_c}\sum_{k=-\frac{N-1}{2}}^{k=\frac{N-1}{2}}T(\beta_c,\beta_b,\Delta\phi_k)=\frac{T(\beta_c,\beta_b)}{N_c}\sum_{k=-\frac{N-1}{2}}^{k=\frac{N-1}{2}}\cos(k\Delta\phi), \qquad \text{(A-7)}$$

such that the calculation can be reduced to the solution of the sum on the right side since $T(\beta_c,\beta_b)$ is known from (A-2). Since the transit time factor here is defined for the complete cavity (and it is multiplied with the average cavity field in subsequent stages of the program) it needs to be divided by the number of cells, N_c. The sum can be solved when transformed into a geometrical series. For that (and to simplify the re-

arrangement of the sum terms) it is of advantage to switch to complex exponentials.

$$\sum_{k=-\frac{N-1}{2}}^{k=\frac{N-1}{2}}\cos(k\Delta\phi)=\frac{1}{2}\sum_{k=-\frac{N-1}{2}}^{k=\frac{N-1}{2}}\left(e^{ik\Delta\phi}+e^{-ik\Delta\phi}\right)=\frac{1}{2}\sum_{k'=0}^{N-1}\left(e^{i\left(k'-\frac{N-1}{2}\right)\Delta\phi}+e^{-i\left(k'-\frac{N-1}{2}\right)\Delta\phi}\right)=$$

$$=\frac{1}{2}e^{-i\frac{N-1}{2}\Delta\phi}\sum_{k'=0}^{N-1}e^{ik'\Delta\phi}+\frac{1}{2}e^{i\frac{N-1}{2}\Delta\phi}\sum_{k'=0}^{N-1}e^{-ik'\Delta\phi}=\left|\sum_{n=0}^{N-1}q^n=\frac{q^N-1}{q-1}\right|=$$

$$=\frac{1}{2}e^{-i\frac{N-1}{2}\Delta\phi}\left(\frac{e^{iN\Delta\phi}-1}{e^{i\Delta\phi}-1}\right)+\frac{1}{2}e^{i\frac{N-1}{2}\Delta\phi}\left(\frac{e^{-iN\Delta\phi}-1}{e^{-i\Delta\phi}-1}\right)=\frac{1}{2}\left(\frac{e^{i\frac{N\Delta\phi}{2}}-e^{-i\frac{N\Delta\phi}{2}}}{e^{i\frac{\Delta\phi}{2}}-e^{-i\frac{\Delta\phi}{2}}}\right)+\frac{1}{2}\left(\frac{e^{-i\frac{N\Delta\phi}{2}}-e^{i\frac{N\Delta\phi}{2}}}{e^{-i\frac{\Delta\phi}{2}}-e^{i\frac{\Delta\phi}{2}}}\right)=$$

$$=\frac{\sin\left(\frac{N\Delta\phi}{2}\right)}{\sin\left(\frac{\Delta\phi}{2}\right)}=\frac{\sin\left[\frac{N_c\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}{\sin\left[\frac{\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}$$

(A-8)

(A-8) can now be multiplied with (A-2) or (A-3) to give the total transit time factor:

$$T(\beta_b,\beta_c,N_c)=\frac{2\cos\left(\frac{\pi}{2}\frac{\beta_c}{\beta_b}\right)}{\pi\left(1-\left(\frac{\beta_c}{\beta_b}\right)^2\right)}\frac{\sin\left[\frac{N_c\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}{N_c\sin\left[\frac{\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}=$$

$$=\frac{\left[1-\left(\frac{\pi^2}{24}\right)\left(1-\frac{\beta_c}{\beta_b}\right)^2+\left(\frac{\pi^4}{1920}\right)\left(1-\frac{\beta_c}{\beta_b}\right)^4\right]}{\left(1+\frac{\beta_c}{\beta_b}\right)}\frac{\sin\left[\frac{N_c\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}{N_c\sin\left[\frac{\pi}{2}\left(1-\frac{\beta_c}{\beta_b}\right)\right]}$$

(A-9)

The transit time factor according to (A-9) is the total transit time factor of the cavity and it assumes that the cavities operate in the π-mode and that the cavity cell length corresponds exactly to half the RF wavelength. The second term which contains the multi-cell effect also has a singularity at $\beta_c=\beta_b$. This singularity needs to be addressed in the program.

Figure 6-2 is a plot of the multi-cell cavity transit factor. It clearly shows that the introduction of many cells strongly reduces the operational window in terms of beam beta. The single cell transit time factor can be reached only in the case $\beta_c=\beta_b$.

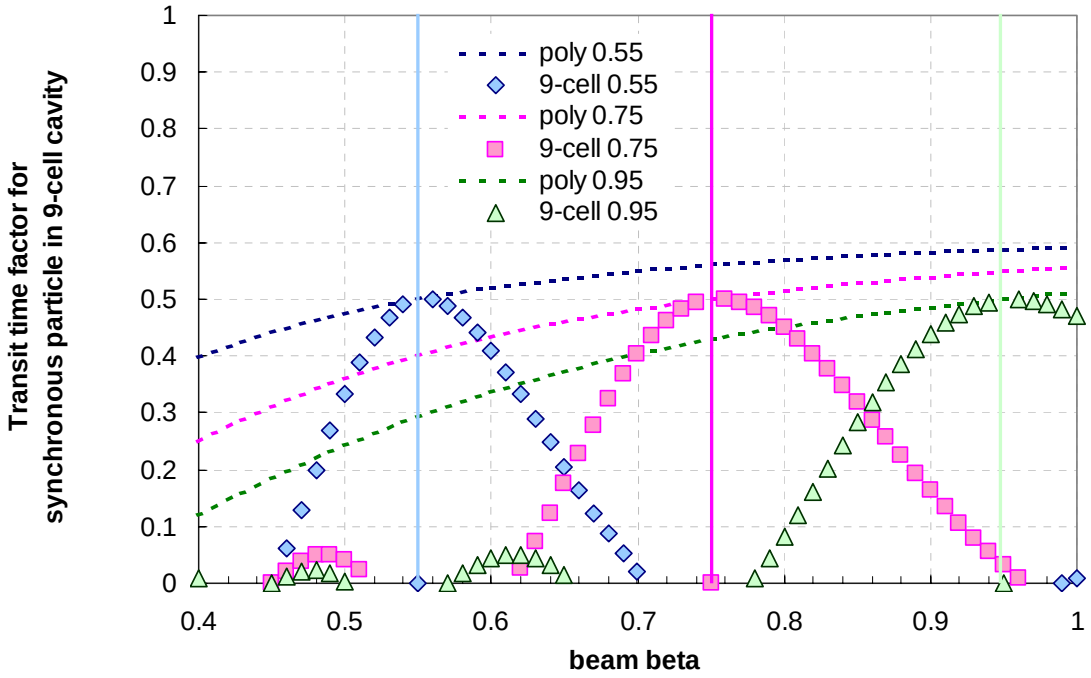


Figure 6-2: Ideal transit time factor for the synchronous particle of varying beta in 9-cell cavities with different betas according to equ. (A-9) The single cell transit time factor is shown as well (dashed lines).

Note that the calculation of the transit time factor introduced above also supposes that the axial field profile in the cavity cell is sinusoidal. Especially in low beta cavities this is not necessarily the case. The program therefore uses a trick to take into account variations *in the cavity designs across the range of betas. In the case of* $\beta=1$ cavities the above assumptions regarding the cavity shape and a sinusoidal field profile hold. If normalized to one (dividing the above term by ½) for the case of $\beta_c=\beta_b=1$, the transit time factor can be applied also to the calculation of the total effective accelerating voltage per cavity in cavities of different designs if it is multiplied to the average accelerating voltage across the cell as calculated by electromagnetic design programs. With this approach the relative transit time factor is used to describe the effect of the mismatch between cavity β and beam β, while the information on the particular field profile is included in the average voltage (that a perfectly matched particle would see) as

obtained, for instance, from a FE model calculation of the particular cavity design.

The gap-factor (L_c/λ - L_c the cell length, λ the RF wave-length) is not considered in Eq.(A-9) because it is implicit in the above calculation of the TTF ($L_c/\lambda = 1/2$)).

The total effective accelerating voltage seen by the particle in a cell of an accelerating cavity is then

$$V_{eff} = \overline{V} \cdot \tilde{T}(\beta_c, \beta_b, N_c) \cdot \cos\phi(\ldots) \quad \overline{V} = \frac{1}{L_c} \int_{-L_c/2}^{L_c/2} V(z) dz \quad (V) \qquad , \qquad \text{(A-10)}$$

where the transit time factor $\tilde{T}$ is given by (A-9) (normalized by multiplication with 2), the average accelerating voltage $\overline{V}$ per cell is obtained from electromagnetic cavity design codes and ϕ is the relative phase of the particle to the phase of the synchronous particle. The normalized transit time factor, $\tilde{T}$, is calculated in the program input sheet.

The program does not include dynamic effects such as:

- the change of particle speed within a cavity
- beam loading in the cavity

7 APPENDIX B

DERIVATION OF THE CAVITY VOLTAGE

Figure 6-3 shows the equivalent circuit of a driven cavity with beam *Ib*=beam current). The schematic indicates possible reflection by the reflective voltage *Ur*. The shunt impedance *Rs* (=*RL*) is also shown. The Kirchhoff-equations to solve for the circuit described in the figure are given next (B-1). The input parameters on the RF power generator side have been transformed to the cavity side.

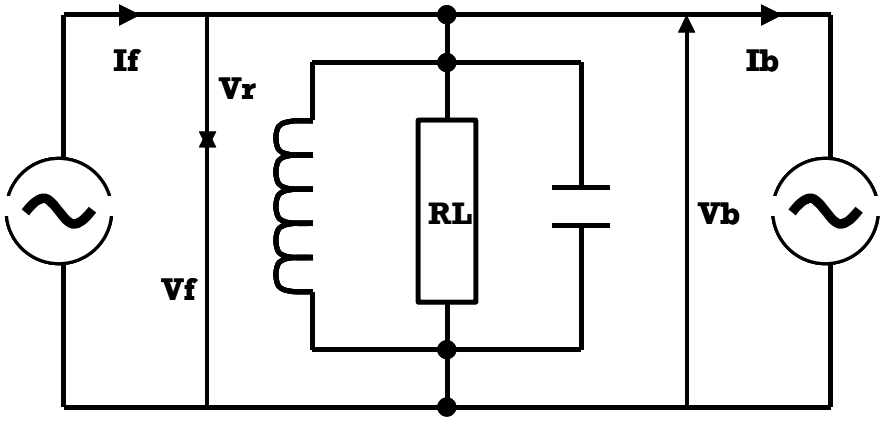


Figure 6-3: Equivalent circuit of cavity operated with beam.

Eq. (B-1) gives the voltage across the cavity, where *RL* is the sum of the coupling losses and the wall losses in the cavity. To obtain optimum coupling with beam, however, cavities need to be over-coupled such that optimum matching occurs in the presence of beam. In the over-coupled case the coupling losses strongly dominate the wall losses: *QL*<<*Q0*, *Rext*>>*R0*. The loaded shunt impedance in the strongly over-coupled limit therefore is $\boldsymbol{R_L \approx R_{ext} = \frac{R}{Q_0} Q_L = rQ_L} \quad (\Omega)$.

$$\boldsymbol{C\ddot{V} + \frac{1}{R_L}\dot{V} + \frac{1}{L}V = \dot{I}_f - \dot{I}_b} \quad \left(\frac{\boldsymbol{A}}{\boldsymbol{s}}\right) \qquad \text{(B-1)}$$

With the half-width $\omega_{1/2}$ and resonance frequency ω_0:

$$\boldsymbol{\omega_{1/2} = \frac{1}{2R_L C} = \frac{\omega_0}{2Q_L}} \qquad \boldsymbol{\omega_0^2 = \frac{1}{LC}} \quad \left(\boldsymbol{rad - Hz}\right) \qquad \text{(B-2)}$$

Eq. (B-1) can be rewritten:

$$\ddot{V} + 2\omega_{1/2}\dot{V} + \omega_0^2 V = R_L \omega_{1/2}\left(2\dot{I}_f - \dot{I}_b\right) \quad \left(\frac{V}{s^2}\right) \tag{B-1)'}$$

In Eq. (B-1)' the additional assumption was made that the forward current is half the generator current. This factor takes into account the total reflection occurring at the cavity input when the cavity is not matched (which is typically the case when there is no beam). This factor 2 essentially takes into account that the voltage at the cavity entrance is two times the desired voltage, as a result of almost total reflection at the input. The almost negligible fraction of power that enters the unmatched cavity and fills it will ultimately establish this double voltage in the cavity.

The voltage can be approximated with Eq. (B-3), as being composed of a high frequency component e^{iwt} and an envelope component that slowly varies in time. Eq. (B-3) assumes a general cavity resonance frequency, ω, that could very well be different from ω_0.

$$V = \hat{V}(t)e^{i\omega t} \qquad (V) \tag{B-3}$$

The same applies to the currents. Inserting the complex voltages into (B-1)', and assuming $\ddot{\hat{V}}, \omega_{1/2}\dot{\hat{V}} \sim 0$ (the voltage varies only very slowly, the cavity bandwidth is very small) gives Eq. (B-4). (B-4) also assumes that the time derivatives of the current are zero. Strictly speaking this assumption is only valid in steady state, long after RF power was switched on.

$$\dot{\hat{V}} + \omega_{1/2}\hat{V} + \frac{\omega_0^2 - \omega^2}{2i\omega}\hat{V} = R_L \omega_{1/2}\left(2\hat{I}_f - \hat{I}_b\right) \quad \left(\frac{V}{s}\right) \tag{B-4}$$

Defining the detuning frequency $\Delta\omega=\omega_0-\omega$ and assuming $\Delta\omega$ to be small, Eq. (B-4) becomes:

$$\dot{\hat{V}} = -\left(\omega_{1/2} - i\Delta\omega\right)\hat{V} + R_L \omega_{1/2}\left(2\hat{I}_f - \hat{I}_b\right) \quad \left(\frac{V}{s}\right) \tag{B-5}$$

This can also be written in a more compact form, where the current term includes forward and beam current:

$$\dot{\hat{V}} = -\left(\omega 12 - i\Delta\omega\right)\hat{V} + \frac{1}{2} r\omega RF\hat{I} \quad \left(\frac{MV}{\mathbf{sec}}\right) \tag{B-5b}$$

Eq. (B-5) is the basic equation describing the field changes in the cavity, which are slow compared to the RF field. The solution of Eq. (B-5) is at the basis of the cavity voltage calculation used in S.C.R.E.A.M.

The filling function, for instance, can be calculated from (B-5) setting the beam current to zero. Building a solution from the homogenous and inhomogeneous version of (B-5), one obtains

$$\hat{V}(t) = \frac{2R_L \hat{I}_f}{\left(1 - i\frac{\Delta\omega}{\omega_{1/2}}\right)}\left[1 - e^{-(\omega_{1/2} - i\Delta\omega)t}\right] \quad (V) \tag{B-6}$$

where the first term represents the to voltage to which the filling process tends asymptotically. In the case $\Delta\omega=0$, the ultimate voltage is $V_{end}=2R_L I_f$. This is the result of complete power reflection at the un-matched cavity! Without detuning, the voltage rises exponentially with the filling time constant $\tau=1/\omega_{1/2}$. In the case in which $\Delta\omega\neq0$ the complex voltage performs rotations in phase-space, spiraling toward the end value $2R_L I_f(1-(\Delta\omega/\omega_{1/2})^2)$ which is always smaller than the ideal value. The detuning factor in the denominator, which describes the reactive component of the cavity impedance in the detuned case, is responsible for this.

Instead of the forward current it is more practical to use the forward power. Forward current and power are related through:

$$P_f = \frac{1}{2}R_{ext}\hat{I}_f^2 \approx \frac{1}{2}R_L\hat{I}_f^2 = \frac{1}{2}\frac{V_c^2}{R_{ext}} = \frac{1}{2}\frac{V_c^2}{R_{beam}} = \frac{1}{2}R_L\hat{I}_b^2 \quad (W) \tag{B-7}$$

when beam (Ib is present).

Instead of solving Eq. B-5 with beam current, the program calculates the beam-loading separately. The voltage reduction, ΔV, in the cavity due to the bunch wakefield can be derived from an energy balance.

$$U_{tot}^{before} = U_{tot}^{after}, \quad \frac{V_0^2}{2r\omega} = \frac{(V_0 - \Delta V)^2}{2r\omega} + V_0 q \quad \Rightarrow \quad \Delta V \approx r\omega I_b \Delta t \tag{B-8}$$

Eq. (B-8) assumes that the beam loading effect is small ($\Delta V << V_0$), so that $O(\Delta V^2)$ terms can be omitted and the $V_0 q$ is approximately the energy taken out from the cavity by the beam. The bunch charge was replaced by $q=I_b\Delta t$ in the final expression. The beam-loading

expression in Eq. (4-12) also includes the transit time factor T' (<1), which actually reduces the cavity V_0 as well as ΔV.

8 APPENDIX C

Scream.m

```
% SCREAM
% SuperConducting RElativistic particle Accelerator siMulation
% written 2003 by M. Huening (mhuening@fnal.gov)
% version 0.1
%

if ~exist('debugging','var'), debugging=0;end

LoadInput;
PreRun;
if debugging==1, return;end

SimCav          = Cavities;
SimCav.KLorentz =
SimCav.KLorentz.*(1+SimCav.Kspread.*randn(size(SimCav.Kspread)));


SimCav.dw       = SimCav.dw-2*pi*SimCav.Amplitude.^2.*(SimCav.KLorentz-
Cavities.KLorentz);
[cavpre,bmpre]  = SimulateField(SimCav,Mod,Bunches,General,Phaseloop);
if debugging==2, return;end

save([datadirectory,'/preresults'],'prerun','cavpre','bmpre','Cavities'
,'Mod','Bunches','General');
if debugging==3, return;end

block_cav_save_count=0;

for kf=1:Nfiles
  SimCav          = Cavities;
  SimCav.KLorentz =
SimCav.KLorentz.*(1+SimCav.Kspread.*randn(size(SimCav.Kspread)));
  SimCav.dw       = SimCav.dw-
2*pi*SimCav.Amplitude.^2.*(SimCav.KLorentz-Cavities.KLorentz);
  for kr=1:Nruns

[cavresult(kr),beamresult(kr)]=SimulateField(SimCav,Mod,Bunches,General
,Phaseloop);
  end

save([datadirectory,'/beamresults',num2str(kf,'%03d')],'beamresult','Si
mCav');
  block_cav_save_count=block_cav_save_count+1;
  if block_cav_save_count>block_cav_save,
    save([datadirectory,'/cavresults',num2str(kf,'%03d')],'cavresult');
    block_cav_save_count=0;
  end
end

cr=cavresult;
br=beamresult;
```

LoadInput.m

```
% Proton Driver Simulation Load Input File
% Written by M. Huening
% mhuening@fnal.gov
%

if ~exist('datadirectory','var'), datadirectory='run0data';end
if ~exist('linacfile','var'), linacfile='linac.csv';end
fid     = fopen([datadirectory,'/',linacfile],'r');

while 1,

 tline = fgetl(fid);
 % end of file reached
 if tline==-1, break;end
 % omit quotes
 tline = tline(tline~='"');
 % parse arrays
 if tline(1) == '{',

    varname    = tline(2:min(find(tline==','))-1);
   fieldnames = fgetl(fid);
   % end of file reached
   if fieldnames==-1,break;end
   % no quotes but add ',' at the end (for easier analysis
   fieldnames = [fieldnames(fieldnames~='"'),','];
   % discard comments
   if any(fieldnames=='%')
      fieldnames=fieldnames(1:min(find(fieldnames=='%'))-1);
      end
   % find the commas
   commas=find(fieldnames==',');
   % take only those who actually separate something
   commas=commas(1:max(find(diff(commas)>1))+1);
   lc=1;
   fieldcell=cell(length(commas),1);
   for kc=1:length(commas)
     fieldcell{kc}=fieldnames(lc:commas(kc)-1);
     lc=commas(kc)+1;
     end
   tline=fgetl(fid);
   A=[];
   while (tline(1)~='}')&(tline~=-1),
     A=[A;sscanf(tline,'%f,')'];
     tline=fgetl(fid);
     tline=tline(tline~='"');
     end
   tmpstruct=struct('name',varname);
   for kc=1:length(fieldcell)
     tmpstruct=setfield(tmpstruct,fieldcell{kc},A(:,kc));
     end
   eval([varname,'=tmpstruct;']);
 else % execute command in tline
   eval(tline);
   end
```

```
 end;
 fclose(fid);

NMod =    max(Cavities.Module);
NCav = length(Cavities.Module);

%Mod=struct([]); Had to comment that out, otherwise it wouldn't run, PB
0305
for k=NMod:-1:1
  Mod(k).Cavities = find(Cavities.Module(:)==k)';
  Mod(k).N        = length(Mod(k).Cavities);
  Mod(k).Feedback = mean(Cavities.Feedback(Cavities.Module==k));
end

if isfield(Cavities,'FillOff'),
  for k=1:NMod
    Mod(k).FillOff = mean(Cavities.FillOff(Mod(k).Cavities));
  end
end

if exist('Phaseloop','var'),
  if isstruct(Phaseloop),
     General.doPhaseloop=any(Phaseloop.Gain);
   end
else
   Phaseloop=[];
   General.doPhaseloop=false;
end;

Cavities.Phase  = Cavities.Phase*pi/180;
Cavities.Module = int32(Cavities.Module);


Cavities.Feedback=Cavities.Feedback...
  ./cellfun('length',{Mod(Cavities.Module).Cavities})';

if ~isfield(Cavities,'KLorentz'),
  Cavities.KLorentz = -1*ones(size(Cavities.Frequency));end
if ~isfield(Cavities,'Kspread'),
  Cavities.Kspread = 0.1*ones(size(Cavities.Frequency));end
if ~isfield(Cavities,'Atten'),
  Cavities.Atten = ones(size(Cavities.Amplitude));end
if ~isfield(Cavities,'FillOff'),
  Cavities.FillOff = zeros(size(Cav.Amplitude));end
if ~isfield(Cavities,'FillTau'),
  Cavities.FillTau=Cavities.Qloaded./Cavities.Frequency/pi;end
if ~isfield(Cavities,'FillTaylor'),
  Cavities.FillTaylor = ones(size(Cavities.Amplitude));end
if ~isfield(Cavities,'ReactiveAmp'),
  Cavities.ReactiveAmp = zeros(size(Cavities.Amplitude));end
if ~isfield(Cavities,'ReactivePhase'),
  Cavities.Reactive = Cavities.ReactiveAmp;
else
  Cavities.Reactive =
Cavities.ReactiveAmp.*exp(i*Cavities.ReactivePhase);
end
```

```
if ~exist('block_cav_save','var'),
  block_cav_save=0;end
```

Prerun.m

```
Pilot.Energy = Bunches.Energy(1);
Pilot.Time = 0;
Pilot.Mass = Bunches.Mass(1);
Pilot.Charge = Bunches.Charge(1);
Pilot.N=1;
Pilot.I=1;

ar=acceleration(Cavities,Pilot,Cavities.Amplitude);
Cavities.Time    = ar.Time;
Cavities.Egain   = diff([Pilot.Energy;ar.Energy(:)]);
Cavities.TTF     = ar.TTF;

prerun = acceleration(Cavities,Bunches,Cavities.Amplitude);

prerun.dt = prerun.Time-prerun.Time(:,1)*ones(1,size(prerun.Time,2));
prerun.de = prerun.Energy-
prerun.Energy(:,1)*ones(1,size(prerun.Energy,2));
```

Acceleration.c

```
#include <mex.h>
#include <math.h>

break
const double MH_PI    = 3.1415296;
const double MH_CVAC  = 2.9979e8;

/* transit time factor */
double ttf(double beta, int n);

double *Beta, *GapLambda, *NCells, *Mode;

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
  double *Position, *BeamTime, *Fieldr, *Fieldi;
  double *PhSync, *Frequency;
  double *Energy, *Time, *TTF;
  double *Energy0,*Time0;
  double *Mass, *Charge;

  mxArray *FieldTmp;
  mxArray *PhaseIn, *AmplIn, *EnergyIn, *TimeIn;
  mxArray *EnergyField, *TimeField, *TTFField;

  int M, N, MM, NN;
  int Ncav, Nbunch, Nrun;
  int k, ncav, nbunch;
```

```
Position  = NULL;
BeamTime  = NULL;

Beta      = NULL;
GapLambda = NULL;
NCells    = NULL;
Mode      = NULL;

Energy    = NULL;
Time      = NULL;


if(nrhs !=3)
  mexErrMsgTxt("Wrong number of inputs!");
if(nlhs >1)
  mexErrMsgTxt("Wrong number of outputs!");
if(!mxIsStruct(prhs[0]))
  mexErrMsgTxt("Input 0 has to be struct!");
if(!mxIsStruct(prhs[1]))
  mexErrMsgTxt("Input 1 has to be struct!");

FieldTmp = mxGetField(prhs[0],0,"Position");
if(!FieldTmp)
  mexErrMsgTxt("Error reading field >>Position<<");
M = mxGetM(FieldTmp);
N = mxGetN(FieldTmp);
Position = (double*)mxGetData(FieldTmp);

FieldTmp = mxGetField(prhs[0],0,"Time");
if(!FieldTmp)
  mexErrMsgTxt("Error reading field >>Time<<");
if(mxGetM(FieldTmp)!=M||mxGetN(FieldTmp)!=N)
  mexErrMsgTxt("All cavity sub fields must have equal size!");
BeamTime = (double*)mxGetData(FieldTmp);

FieldTmp = mxGetField(prhs[0],0,"Frequency");
if(!FieldTmp)
  mexErrMsgTxt("Error reading field >>Frequency<<");
if(mxGetM(FieldTmp)!=M||mxGetN(FieldTmp)!=N)
  mexErrMsgTxt("All cavity sub fields must have equal size!");
Frequency = (double*)mxGetData(FieldTmp);

FieldTmp = mxGetField(prhs[0],0,"Phase");
if(!FieldTmp)
  mexErrMsgTxt("Error reading field >>Phase<<");
if(mxGetM(FieldTmp)!=M||mxGetN(FieldTmp)!=N)
  mexErrMsgTxt("All cavity sub fields must have equal size!");
PhSync   = (double*)mxGetData(FieldTmp);

FieldTmp = mxGetField(prhs[0],0,"Beta");
if(!FieldTmp)
  mexErrMsgTxt("Error reading field >>Beta<<");
if(mxGetM(FieldTmp)!=M||mxGetN(FieldTmp)!=N)
  mexErrMsgTxt("All cavity sub fields must have equal size!");
Beta      = (double*)mxGetData(FieldTmp);
```

```
  FieldTmp = mxGetField(prhs[0],0,"GapLambda");
  if(!FieldTmp)
    mexErrMsgTxt("Error reading field >>GapLambda<<");
  if(mxGetM(FieldTmp)!=M||mxGetN(FieldTmp)!=N)
    mexErrMsgTxt("All cavity sub fields must have equal size!");
  GapLambda = (double*)mxGetData(FieldTmp);

  FieldTmp = mxGetField(prhs[0],0,"Cells");
  if(!FieldTmp)
    mexErrMsgTxt("Error reading field >>Cells<<");
  if(mxGetM(FieldTmp)!=M||mxGetN(FieldTmp)!=N)
    mexErrMsgTxt("All cavity sub fields must have equal size!");
  NCells    = (double*)mxGetData(FieldTmp);

  FieldTmp = mxGetField(prhs[0],0,"Mode");
  if(!FieldTmp)
    mexErrMsgTxt("Error reading field >>Mode<<");
  if(mxGetM(FieldTmp)!=M||mxGetN(FieldTmp)!=N)
    mexErrMsgTxt("All cavity sub fields must have equal size!");
  Mode      = (double*)mxGetData(FieldTmp);

  if(mxGetM(prhs[2])!=M||mxGetN(prhs[2])!=N)
    mexErrMsgTxt("Field information must have same size as Cavity
Fields!");
  Fieldr = mxGetPr(prhs[2]);
  if(!mxIsComplex(prhs[2]))
    Fieldi=mxGetPr(mxCreateDoubleMatrix(M,N,mxREAL));
  else
    Fieldi = mxGetPi(prhs[2]);
  if(!Fieldi) mexErrMsgTxt("Oops!");

  if(M>N){
    if(N>1)
      mexErrMsgTxt("cavity data may only have one dimension");
    Ncav = M;
  }
  else{
    if(M>1)
      mexErrMsgTxt("cavity data may only have one dimension");
    Ncav = N;
  }

  EnergyIn = mxGetField(prhs[1],0,"Energy");
  if(!EnergyIn)
    mexErrMsgTxt("Error reading field >>Energy<<");
  MM = mxGetM(EnergyIn);
  NN = mxGetN(EnergyIn);
  Energy0 = (double*)mxGetData(EnergyIn);

  TimeIn = mxGetField(prhs[1],0,"Time");
  if(!TimeIn)
    mexErrMsgTxt("Error reading field >>Time<<");
  if(mxGetM(TimeIn)!=MM||mxGetN(TimeIn)!=NN)
    mexErrMsgTxt("All bunch sub fields must have equal size!");
  Time0   = (double*)mxGetData(TimeIn);

  FieldTmp = mxGetField(prhs[1],0,"Mass");
```

```
if(!FieldTmp)
  mexErrMsgTxt("Error reading field >>Mass<<");
if(mxGetM(FieldTmp)!=MM||mxGetN(FieldTmp)!=NN)
  mexErrMsgTxt("All bunch sub fields must have equal size!");
Mass    = (double*)mxGetData(FieldTmp);

FieldTmp = mxGetField(prhs[1],0,"Charge");
if(!FieldTmp)
  mexErrMsgTxt("Error reading field >>Charge<<");
if(mxGetM(FieldTmp)!=MM||mxGetN(FieldTmp)!=NN)
  mexErrMsgTxt("All bunch sub fields must have equal size!");
Charge   = (double*)mxGetData(FieldTmp);

if(MM>NN){
  if(NN>1)
    mexErrMsgTxt("bunch data may only have one dimension");
  Nbunch  = MM;
}else{
  if(MM>1)
    mexErrMsgTxt("bunch data may only have one dimension");
  Nbunch  = NN;
}

EnergyField = mxCreateDoubleMatrix(Ncav,Nbunch,mxREAL);
TimeField   = mxCreateDoubleMatrix(Ncav,Nbunch,mxREAL);
TTFField    = mxCreateDoubleMatrix(Ncav,Nbunch,mxREAL);
Energy      = (double*)mxGetData(EnergyField);
Time        = (double*)mxGetData(TimeField);
TTF         = (double*)mxGetData(TTFField);

if(!Energy||!Time)
  mexErrMsgTxt("Something went wrong.");

plhs[0] = mxCreateStructMatrix(1,1,0,NULL);
mxAddField(plhs[0],"Energy");
mxSetFieldByNumber(plhs[0],0,0,EnergyField);
mxAddField(plhs[0],"Time");
mxSetFieldByNumber(plhs[0],0,1,TimeField);
mxAddField(plhs[0],"TTF");
mxSetFieldByNumber(plhs[0],0,2,TTFField);

for(nbunch=0;nbunch<Nbunch;nbunch++)
  {
    double ta, ea, pa, ba, ga ,ma, ph;
    pa = 0;                  /* bunch position */
    ta = Time0[nbunch];      /* arrival time */
    ea = Energy0[nbunch];    /* actual energy */
    ma = Mass[nbunch];       /* particle mass */
    ga = 1+ea/ma;            /* gamma */
    ba = sqrt(1-1/ga/ga);    /* beta */
    for(ncav=0;ncav<Ncav;ncav++)
 {
   double de;
   /* calculate arrival time */
   ta = ta+(Position[ncav]-pa)/ba/MH_CVAC;
   pa = Position[ncav];
   Time[nbunch*Ncav+ncav] = ta;
```

```
    /* convert into phase difference */
    if(BeamTime[ncav]<0)
      ph = 0;
    else
      ph = 2*MH_PI*(BeamTime[ncav]-ta)*Frequency[ncav];
    /* calculate energy gain */
    TTF[nbunch*Ncav+ncav]=ttf(ba,ncav);
    de = Charge[nbunch]*TTF[nbunch*Ncav+ncav]
       *(Fieldr[ncav]*cos(PhSync[ncav]-ph)
        -Fieldi[ncav]*sin(PhSync[ncav]-ph));

    ea = ea+de;
    Energy[nbunch*Ncav+ncav] = ea;
    ga = 1+ea/ma;
    ba = sqrt(1-1/ga/ga);
  }
   }
}



double ttf(double beta, int n)
{
  double x, y, yy, cells, gap;
  y     = Beta[n]/(beta+1e-10);
  yy    = MH_PI*(1-y)*GapLambda[n];
  gap   = (1-yy*yy/6+yy*yy*yy*yy/120)/(1+y)/GapLambda[n];
  x     = Mode[n]*(1-y);
  cells = sin(NCells[n]*x/2)/(NCells[n]*sin(x/2));
  return gap*cells;
}
```

SimulateField.m

```
function [cr,br,randstate] =
SimulateField(Cav,Mod,Bunch,General,Phloop,randstate)

if nargin>5, randn('state',randstate);end
br.randstate = randn('state');

NFill  = General.Filltime/General.Stepsize;
NBeam  = General.Beamtime/General.Stepsize;
NStep  = NFill+NBeam;%+100;

br.Eoff = randn*General.Ecoherent;
br.Toff = randn*General.Tcoherent;
br.Ioff = randn*General.Icoherent;
br.Efluc = randn(NBeam,1)*General.Efluc;
br.Tfluc = randn(NBeam,1)*General.Tfluc;
br.Ifluc = randn(NBeam,1)*General.Ifluc;

Bunch.Time   = Bunch.Time+br.Toff;
Bunch.Energy = Bunch.Energy+br.Eoff;
Bunch.I      = Bunch.I*(1+br.Ioff/100);
```

```
Bun = Bunch;

dt  = General.Stepsize*1e-6;

NCav   = length(Cav.Amplitude);
NMod   = length(Mod);
NBun   = length(Bun.N);

cr.sh1    = zeros(NCav,NStep);
cr.sh2    = zeros(NCav,NStep);
cr.ps1    = zeros(NCav,NStep);
cr.ps2    = zeros(NCav,NStep);


cr.CField = zeros(NCav,NStep);
cr.CForwd = zeros(NMod,NStep);
cr.CFwdpl = ones(NCav,NStep);
cr.CRvspl = zeros(NCav,NStep);
cr.CDrive = zeros(NCav,NStep);


cr.SField = zeros(NCav,NStep);
cr.SForwd = zeros(NMod,NStep);

cr.CCur   = zeros(NCav,NBeam);
cr.ECur   = zeros(NCav,NBeam);

br.Energy = zeros(NBun,NBeam);
br.Time   = zeros(NBun,NBeam);

if (nargin<5), Phloop=[];end
if isempty(Phloop),
  General.doPhaseloop=0;
  PLIdx = [];
else
  General.doPhaseloop=1;
  initphaseloop;
end

NPhl   = length(PLIdx);

simbeam   = zeros(NCav,1);

% Lorentz Detuning Constant (has some fluctuation to it)
cr.K      = -abs(Cav.KLorentz);
% Predetuning to compensate Lorentz Force
dwpre     = Cav.dw;%-2*pi*Cav.Amplitude.^2.*(cr.K-Cav.KLorentz);
% Where the Lorentz Force will be stored
dwlor     = 0*dwpre;
% Microphonics Coherent (0 Hz)
dwmic0    = dwpre+2*pi*Cav.Microphonics.*randn(NCav,1);
cr.w12    = pi*Cav.Frequency./Cav.Qloaded;
cr.dw     = zeros(NCav,NStep);
RL        = Cav.Qloaded.*Cav.Rshunt;
```

```
if ~isfield(Cav,'Attenuation'),
  Cav.Attenuation = ones(size(Cav.Amplitude));
end
Conversion     = sqrt(2*RL);
Cav.Feedback   = Cav.Feedback./Conversion./Cav.Attenuation;

for km = 1:size(cr.SField,1)
  NOff=Cav.FillOff(km);
  NTau=Cav.FillTau(km)/dt;
  nTau=((0:(NFill-NOff)))/NTau;
  cr.SField(km,1+NOff:NFill+1) = 1-exp(-nTau)+Cav.FillTaylor(km)*(nTau-
1+exp(-nTau));
  cr.SField(km,1+NOff:NFill+1) =
cr.SField(km,1+NOff:NFill+1)/cr.SField(km,NFill+1)*Cav.Amplitude(km);
  cr.SField(km,NFill+1:end) = Cav.Amplitude(km);
end


IFac = 2*pi*Cav.Rshunt.*Cav.Frequency.*exp(-i*Cav.Phase)*dt*1e-6;

for km=1:length(Mod)
  AIFac = mean(IFac(Mod(km).Cavities)...
              ./(1-exp(-cr.w12(Mod(km).Cavities)*dt))...
              ./Conversion(Mod(km).Cavities));
  cr.SForwd(km,NFill+1:NFill+NBeam) = AIFac*sum(Bunch.I);
  cr.SForwd(km,1:NFill)             =
mean(Cav.Amplitude(Mod(km).Cavities)./Conversion(Mod(km).Cavities))/2;
  cr.Sforwd(km,1:Cav.FillOff(Mod(km).Cavities(1)))=0;
end

ct0 = Cav.Time*ones(size(Bun.N'));
cf0 = Cav.Frequency*ones(size(Bun.N'));
E0  = cumsum(Cav.Egain);



for ks=1:(NFill+NBeam)
  kb=ks-NFill;
  cr.dw(:,ks) = dwmic0+dwlor+2*pi*Cav.FastMicrophonics.*randn(NCav,1);
  if ks>NFill,
    Bun.Time       = Bunch.Time+br.Tfluc(kb);
    Bun.Energy     = Bunch.Energy+br.Efluc(kb);
    Bun.I          = Bunch.I*(1+br.Ifluc(kb)/100);
    ar             = acceleration(Cav,Bun,cr.CField(:,ks));
    cr.ECur(:,kb)  = (ar.Energy*Bun.N)/sum(Bun.N)-E0;
    cr.CCur(:,kb)  = (exp(2i*pi*(ct0-ar.Time).*cf0).*ar.TTF)*Bun.I;
    br.Energy(:,kb)= ar.Energy(end,:)';
    br.Time(:,kb)  = ar.Time(end,:)';
    simbeam        = IFac.*cr.CCur(:,kb);
    end
  if ks<(NFill+NBeam)

    cr.CForwd(:,ks+0) = dimsum((cr.SField(:,ks)-
cr.CField(:,ks)).*Cav.Feedback,...
                               Cav.Module,NMod)+cr.SForwd(:,ks);
    % phase shifter and conversion to Ue
```

```
    Drive             =
cr.CForwd(Cav.Module,ks).*cr.CFwdpl(:,ks).*Conversion.*Cav.Attenuation;
    cr.CDrive(:,ks+1) = Drive+(cr.CField(:,ks)-
Drive).*cr.CRvspl(:,ks)./(1-cr.CRvspl(:,ks));
    % cavity dynamics
    CP                = 2*cr.w12.*cr.CDrive(:,ks+1)./(cr.w12-
i*cr.dw(:,ks));
    cr.CField(:,ks+1) = (cr.CField(:,ks)-simbeam-CP).*exp(-(cr.w12-
i*cr.dw(:,ks))*dt)+CP;
    % Lorentz-Force Detuning
    dwlor  = detuning(cr.CField(:,ks),dwlor,dt,cr.K);
    if General.doPhaseloop,
      dophaseloop;
      %dophaseloop_ideal;
    end
    %end of ks<NFill+NBeam
  end
  %end of for ks
end

if isfield(General,'Downsample'),
  cr.CField = cr.CField(:,1:General.Downsample:end);
  cr.SField = cr.SField(:,1:General.Downsample:end);
  cr.CForwd = cr.CForwd(:,1:General.Downsample:end);
  cr.SForwd = cr.SForwd(:,1:General.Downsample:end);
  cr.CFwdpl = cr.CFwdpl(:,1:General.Downsample:end);
  cr.CCur   = cr.CCur(:,1:General.Downsample:end);
  cr.ECur   = cr.ECur(:,1:General.Downsample:end);
  cr.dw     = cr.dw(:,1:General.Downsample:end);
  end
```

Detuning.m

```
function dwn=detuning(field,dwo,dt,K);

if nargin<4, K=-1.0;end
tau=3e-4;

dwn=dwo-dt./tau.*dwo+2*pi*dt./tau.*K.*abs(field).^2;
```

Dimsum.c

```
#include <mex.h>
#include <math.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
  int NMod, NCav, k;
  double *IPr, *IPi;
  double *SPr, *SPi;
  int32_T *idx;
  mxArray *FieldTmp;
```

```
  if(!mxIsInt32(prhs[1]))
    mexErrMsgTxt("Index Values must be 32 Bit Integer!");
  idx = (int32_T*)mxGetData(prhs[1]);

  NMod = (int)mxGetScalar(prhs[2]);
  NCav = mxGetNumberOfElements(prhs[0]);

  if(mxIsComplex(prhs[0])){
    FieldTmp = mxCreateDoubleMatrix(NMod,1,mxCOMPLEX);
    SPr = mxGetPr(FieldTmp)-1;
    SPi = mxGetPi(FieldTmp)-1;
    IPr = mxGetPr(prhs[0]);
    IPi = mxGetPi(prhs[0]);
  }else{
    FieldTmp = mxCreateDoubleMatrix(NMod,1,mxREAL);
    SPr = mxGetPr(FieldTmp)-1;
    IPr = mxGetPr(prhs[0]);
  }
  plhs[0] = FieldTmp;

  for(k=0;k<NCav;k++){
    SPr[idx[k]] = SPr[idx[k]]+IPr[k];
  }
  if(mxIsComplex(prhs[0]))
    for(k=0;k<NCav;k++){
      SPi[idx[k]] = SPi[idx[k]]+IPi[k];
    }
}
```

initphaseloop.m

```
%Parameter initialization for differential/proportional phaseshifter

PLIdx  = Phloop.CavNo;
ALG    = Phloop.AmpGain;
PLG    = Phloop.Gain;
PLTau  = General.PhaseTau;
ALD    = Phloop.AmpDGain;
PLD    = Phloop.DGain;
PLdel  = 0;

PLSat  = pi/4;
PLSatn = 1*2*PLSat/pi;
PLIni  = 1*pi/4;
PLsin  = sin(PLIni);
PLIna  = cos(PLIni);
ALSat  = 1.2;


psh1 = 0;
psh2 = 0;
cr.sh1(PLIdx,1) = psh1;
cr.sh2(PLIdx,1) = psh2;
```

dophaseloop.m

```
%simulates proportional-differential phase-shifter

afld = cr.SField(:,ks)-cr.CField(:,ks);
mfld = dimsum(afld,Cav.Module,length(Mod))./[Mod(:).N]';

dfld = afld(PLIdx)-mfld(Cav.Module(PLIdx));

if ks==1
   ofld = dfld;
end;

damp = real(dfld).*ALG+real(dfld-ofld).*ALD;
dpha = imag(dfld).*PLG+imag(dfld-ofld).*PLD;

dsh1 = PLSatn*atan((dpha+damp/PLsin)/PLSatn);
dsh2 = PLSatn*atan((dpha-damp/PLsin)/PLSatn);
psh1 = psh1+dsh1/PLTau;
psh2 = psh2+dsh2/PLTau;
cr.sh1(PLIdx,ks+1) = psh1;
cr.sh2(PLIdx,ks+1) = psh2;

if ks>PLdel,
   cr.CFwdpl(PLIdx,ks+1)=...
   (exp(i*(cr.sh1(PLIdx,ks-PLdel)-PLIni))+...
    exp(i*(cr.sh2(PLIdx,ks-PLdel)+PLIni)))...
    /PLIna/2;
   cr.CRvspl(PLIdx,ks+1)=Cav.Reactive(PLIdx).*sin(psh1).*exp(i*psh2);
end;

ofld = dfld;
```